TEXAS INSTRUMENTS

TMS320C62xx Peripherals

Reference Guide

TI DSP SOLUTIONS

15 YEARS OF LEADERSHIP

1997 Digital Signal Processing Solutions

Texas Instruments

Printed in U.S.A., October 1997

SPRU190A

Reference Guide

TMS320C62xx Peripherals

1997

TMS320C62xx Peripherals Reference Guide

Literature Number: SPRU190A
October 1997

Printed on Recycled Paper

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Preface

Read This First

About This Manual

This reference guide describes the on-chip peripherals of the TMS320C62xx digital signal processors (DSPs). Main topics are the program memory, data memory, direct memory access (DMA) controller, host port, external memory interface (EMIF), boot configuration, multichannel serial ports (MCSPs), timers, interrupt selector and external interrupts, device clocking, and power-down modes.

Notational Conventions

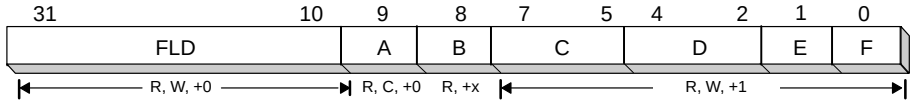
This document uses the following conventions:

- Program listings, program examples, file names, and symbol names are shown in a `special font`. Here is a sample program listing:

```
LDW .D1     *A0,A1
ADD .L1     A1,A2,A3
NOP         3
MPY .M1     A1,A4,A5
```

- Throughout this book MSB means *most significant bit* and LSB means *least significant bit*.

- Register notation:



Legend: R Readable by the instruction
W Writeable by the instruction
+x Value undefined after reset
+0 Value is zero after reset
+1 Value is zero after reset
C Clearable using the **MVC** instruction

Information About Cautions and Warnings

This book may contain cautions and warnings.

This is an example of a caution statement.

A caution statement describes a situation that could potentially damage your software or equipment.

This is an example of a warning statement.

A warning statement describes a situation that could potentially cause harm to you.

The information in a caution or warning is provided for your protection. Please read each caution and warning carefully.

Related Documentation From Texas Instruments

The following books describe the TMS320C6x family and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477–8924. When ordering, please identify the book by its title and literature number.

TMS320C62xx Technical Brief (literature number SPRU197) gives an introduction to the ’C62xx digital signal processor, development tools, and third-party support.

TMS320C62xx CPU and Instruction Set Reference Guide (literature number SPRU189) describes the ’C62xx CPU architecture, instruction set, pipeline, and interrupts for the TMS320C62xx digital signal processors.

TMS320C62xx Programmer’s Guide (literature number SPRU198) describes ways to optimize C and assembly code and includes application program examples.

TMS320C6x Assembly Language Tools User’s Guide (literature number SPRU186) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the ’C6x generation of devices.

TMS320C6x Optimizing C Compiler User’s Guide (literature number SPRU187) describes the ’C6x C compiler. This C compiler accepts ANSI standard C source code and produces assembly language source code for the ’C6x generation of devices. This book also describes the assembly optimizer, which helps you optimize your assembly code.

TMS320C6x C Source Debugger User’s Guide (literature number SPRU188) tells you how to invoke the ’C6x simulator and emulator versions of the C source debugger interface. This book discusses various aspects of the debugger, including command entry, code execution, data management, breakpoints, profiling, and analysis.

TMS320C6201 Digital Signal Processor Data Sheet (literature number SPRS051) describes the features of the TMS320C6xx and provides pinouts, electrical specifications, and timings for the device.

Trademarks

If You Need Assistance . . .

❑ World-Wide Web Sites

TI Online	http://www.ti.com
Semiconductor Product Information Center (PIC)	http://www.ti.com/sc/docs/pic/home.htm
DSP Solutions	http://www.ti.com/dsps
320 Hotline On-line™	http://www.ti.com/sc/docs/dsps/support.htm

❑ North America, South America, Central America

Product Information Center (PIC)	(972) 644-5580		
TI Literature Response Center U.S.A.	(800) 477-8924		
Software Registration/Upgrades	(214) 638-0333	Fax: (214) 638-7742	
U.S.A. Factory Repair/Hardware Upgrades	(281) 274-2285		
U.S. Technical Training Organization	(972) 644-5580		
DSP Hotline	(281) 274-2320	Fax: (281) 274-2324	Email: dsph@ti.com
DSP Modem BBS	(281) 274-2323		

DSP Internet BBS via anonymous ftp to ftp://ftp.ti.com/pub/tms320bbs

❑ Europe, Middle East, Africa

European Product Information Center (EPIC) Hotlines:

Multi-Language Support	+33 1 30 70 11 69	Fax: +33 1 30 70 10 32	Email: epic@ti.com
Deutsch	+49 8161 80 33 11 or +33 1 30 70 11 68		
English	+33 1 30 70 11 65		
Francais	+33 1 30 70 11 64		
Italiano	+33 1 30 70 11 67		
EPIC Modem BBS	+33 1 30 70 11 99		
European Factory Repair	+33 4 93 22 25 40		
Europe Customer Training Helpline		Fax: +49 81 61 80 40 10	

❑ Asia-Pacific

Literature Response Center	+852 2 956 7288	Fax: +852 2 956 2200
Hong Kong DSP Hotline	+852 2 956 7268	Fax: +852 2 956 1002
Korea DSP Hotline	+82 2 551 2804	Fax: +82 2 551 2828
Korea DSP Modem BBS	+82 2 551 2914	
Singapore DSP Hotline		Fax: +65 390 7179
Taiwan DSP Hotline	+886 2 377 1450	Fax: +886 2 377 2718
Taiwan DSP Modem BBS	+886 2 376 2592	

Taiwan DSP Internet BBS via anonymous ftp to ftp://dsp.ee.tit.edu.tw/pub/TI/

❑ Japan

Product Information Center	+0120-81-0026 (in Japan)	Fax: +0120-81-0036 (in Japan)
	+03-3457-0972 or (INTL) 813-3457-0972	Fax: +03-3457-1259 or (INTL) 813-3457-1259
DSP Hotline	+03-3769-8735 or (INTL) 813-3769-8735	Fax: +03-3457-7071 or (INTL) 813-3457-7071
DSP BBS via Nifty-Serve	Type “Go TIASP”	

❑ Documentation

When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number.

Mail: Texas Instruments Incorporated
Technical Documentation Services, MS 702
P.O. Box 1443
Houston, Texas 77251-1443

Note: When calling a Literature Response Center to order documentation, please specify the literature number of the book.

Contents

Figures

Tables

Chapter 1

Introduction

The TMS320C6x generation of digital signal processors is part of the TMS320 family of digital signal processors (DSPs). The TMS320C62xx devices are fixed-point DSPs in the TMS320C6x generation. The TMS320C62xx is the first DSP to use the VelociTI™ architecture, a high-performance, advanced VLIW (very long instruction word) architecture, making the ’C62xx an excellent choice for multichannel, multifunction, and high data rate applications.

The ’C62xx’s VelociTI architecture makes it the first off-the-shelf DSP to use advanced VLIW to achieve high performance through increased instruction-level parallelism. The VelociTI advanced VLIW architecture uses multiple execution units operating in parallel to execute multiple instructions during a single clock cycle. Parallelism is the key to extremely high performance, taking these DSPs well beyond the performance capabilities of traditional designs.

Topic **Page**

1.1 TMS320 Family Overview

The TMS320 family consists of fixed-point, floating-point, and multiprocessor digital signal processors (DSPs). TMS320 DSPs have an architecture designed specifically for real-time signal processing.

1.1.1 History of TMS320 DSPs

In 1982, Texas Instruments introduced the TMS32010—the first fixed-point DSP in the TMS320 family. Before the end of the year, *Electronic Products* magazine awarded the TMS32010 the title “Product of the Year”. Today, the TMS320 family consists of many generations: ’C1x, ’C2x, ’C2xx, ’C5x, and ’C54x fixed-point DSPs; ’C3x and ’C4x floating-point DSPs; and ’C8x multiprocessor DSPs. Now there is a new generation of DSPs, the TMS320C6x generation, with performance and features that are reflective of Texas Instruments commitment to lead the world in DSP solutions.

1.1.2 Typical Applications for the TMS320 Family

Table 1–1 lists some typical applications for the TMS320 family of DSPs. The TMS320 DSPs offer adaptable approaches to traditional signal-processing problems. They also support complex applications that often require multiple operations to be performed simultaneously.

Table 1–1. Typical Applications for the TMS320 DSPs

Automotive	Consumer	Control
Adaptive ride control	Digital radios/TVs	Disk drive control
Antiskid brakes	Educational toys	Engine control
Cellular telephones	Music synthesizers	Laser printer control
Digital radios	Pagers	Motor control
Engine control	Power tools	Robotics control
Global positioning	Radar detectors	Servo control
Navigation	Solid-state answering machines	
Vibration analysis		
Voice commands		
General Purpose	**Graphics/Imaging**	**Industrial**
Adaptive filtering	3-D computing	Numeric control
Convolution	Animation/digital maps	Power-line monitoring
Correlation	Homomorphic processing	Robotics
Digital filtering	Image compression/transmission	Security access
Fast Fourier transforms	Image enhancement	
Hilbert transforms	Pattern recognition	
Waveform generation	Robot vision	
Windowing	Workstations	
Instrumentation	**Medical**	**Military**
Digital filtering	Diagnostic equipment	Image processing
Function generation	Fetal monitoring	Missile guidance
Pattern matching	Hearing aids	Navigation
Phase-locked loops	Patient monitoring	Radar processing
Seismic processing	Prosthetics	Radio frequency modems
Spectrum analysis	Ultrasound equipment	Secure communications
Transient analysis		Sonar processing
Telecommunications		**Voice/Speech**
1200- to 56 600-bps modems	Faxing	Speaker verification
Adaptive equalizers	Future terminals	Speech enhancement
ADPCM transcoders	Line repeaters	Speech recognition
Base stations	Personal communications systems (PCS)	Speech synthesis
Cellular telephones	Personal digital assistants (PDA)	Speech vocoding
Channel multiplexing	Speaker phones	Text-to-speech
Data encryption	Spread spectrum communications	Voice mail
Digital PBXs	Digital subscriber loop (xDSL)	
Digital speech interpolation (DSI)	Video conferencing	
DTMF encoding/decoding	X.25 packet switching	
Echo cancellation		

1.2 Overview of the TMS320C6x Generation of Digital Signal Processors

With a performance of up to 1600 million instructions per second (MIPS) and an efficient C compiler, the TMS320C6x DSPs give system architects unlimited possibilities to differentiate their products. High performance, ease of use, and affordable pricing make the TMS320C6x generation the ideal solution for multichannel, multifunction applications, such as:

- Pooled modems
- Wireless base stations
- Remote access servers (RAS)
- Digital subscriber loop (DSL) systems
- Cable modems
- Multichannel telephony systems

The TMS320C6x generation is also an ideal solution for exciting new applications, for example:

- Personalized home security with face and hand/fingerprint recognition
- Advanced cruise control with GPS navigation and accident avoidance
- Remote medical diagnostics

1.3 Features and Options of the TMS320C62xx

At 200 MHz, the ’C62xx devices operate at a 5-ns cycle time, executing up to eight 32-bit instructions every cycle. The device’s core CPU consists of 32 general-purpose registers of 32-bit word length and eight functional units:

- Two multipliers
- Six ALUs

The ’C62xx has a complete set of optimized development tools, including an efficient C compiler, an assembly optimizer for simplified assembly-language programming and scheduling, and a Windows™ based debugger interface for visibility into source code execution characteristics. A hardware emulation board, compatible with the TI XDS510™ emulator interface, is also available. This tool complies with IEEE Standard 1149.1–1990, IEEE Standard Test Access Port and Boundary Scan Architecture.

Features of the ’C62xx include:

- Advanced VLIW CPU with eight functional units, including two multipliers and six arithmetic units
  - Executes up to eight instructions per cycle for up to ten times the performance of other DSPs
  - Allows designers to develop highly effective RISC-like code for fast development time
- Instruction packing
  - Gives code size equivalence for eight instructions executed serially or in parallel
  - Reduces code size, program fetches, and power consumption
- All instructions execute conditionally
  - Reduces costly branching
  - Increases parallelism for higher sustained performance
- Efficient code execution on independent functional units.
  - Industry’s most efficient C compiler on DSP benchmark suite
  - Industry’s first assembly optimizer for fast development and improved parallelism

- 8/16/32-bit data support, providing efficient memory support for a variety of applications
- 40-bit arithmetic options add extra precision for vocoders and other computationally intensive applications
- Saturation and normalization provide support for key arithmetic operations.
- Field manipulation and instruction extract, set, clear, and bit counting support common operation found in control and data manipulation applications.

For more information on features and options of the TMS320C62xx, see the *TMS320C62xx CPU and Instruction Set Reference Guide*. For more information on the EMIF, see the *TMS320C62xx Peripherals Reference Guide*.

1.4 Overview of TMS320C62xx Memory

The TMS320C62xx memory map consists of:

- Internal program memory
- Internal data memory
- Internal peripherals
- External memory accessed through the EMIF

The internal program memory can be mapped into the CPU address space or operate as a cache. A 256-bit wide path is provided from to the CPU to allow a continuous stream of 8 32-bit instructions for maximum performance. When off-chip memory is used, the external memory interface (EMIF) unifies these spaces to a single memory space on most devices, if so desired.

The data memory controller connects:

- The CPU and direct memory access (DMA) controller to internal data memory and performs the necessary arbitration
- The CPU data access to the external memory interface (EMIF)
- The CPU to on-chip peripherals through the peripheral bus controller

The data memory controller services all requests to internal data memory as well as all CPU data requests to external data memory.

The CPU sends requests to the data memory controller. Store data is transmitted through the CPU data store buses. Load data is received through the CPU data load buses (LD1 and LD2). The CPU data requests are mapped based on the range of the memory address to either the internal data memory, internal peripheral space (through the peripheral bus controller) or the external memory interface. The data memory controller also connects the DMA to the internal data memory and performs CPU/DMA arbitration for the on-chip data RAM.

Both the CPU and DMA can read and write 8-bit bytes, 16-bit halfwords, and 32-bit words. The data memory controller performs arbitration between the CPU and DMA independently for each 16-bit block.

Interleaved memory organization allows the CPU to access two addresses in memory simultaneously.

The CPU and DMA support configurable endianness. This endianness is selected by the LENDIAN (little endian) pin on the device. This selection applies to both the CPU and the DMA controller.

The peripheral bus controller performs arbitration between the CPU and DMA for the on-chip peripherals. The peripherals are controlled by the CPU through

accesses of control registers. The CPU accesses these registers through the peripheral data bus. For data accesses, the DMA controller accesses the peripheral bus controller (PBC) directly, while the CPU accesses the PBC through the data memory controller.

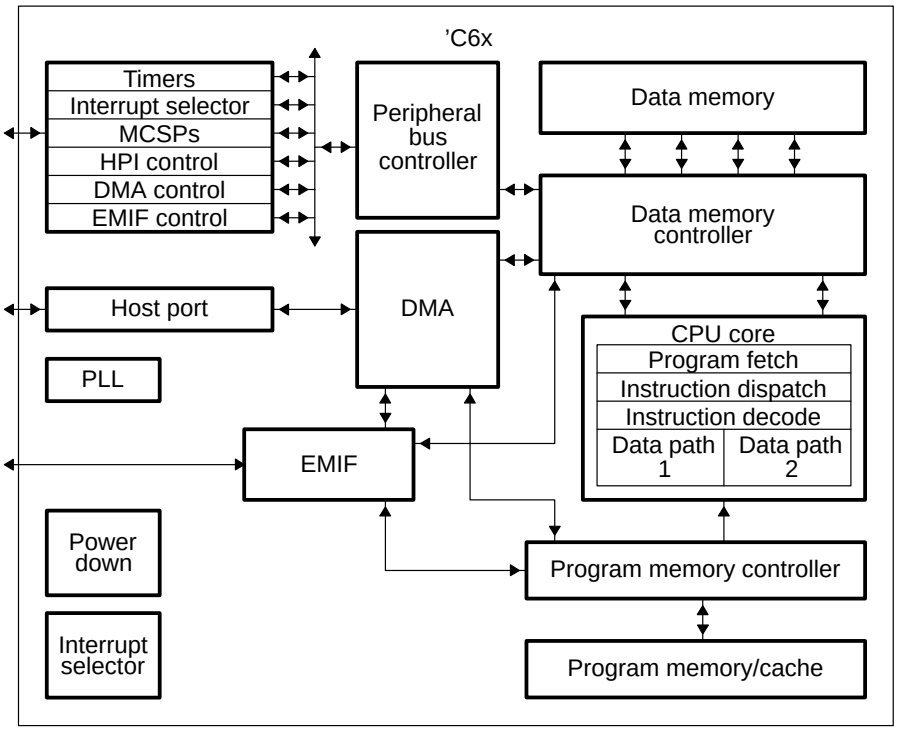
1.5 Overview of TMS320C62xx Peripherals

Peripherals for the TMS320C62xx devices include:

- Direct memory access (DMA) controller
- Host-port interface (HPI)
- External memory interface (EMIF)
- Boot configuration
- Multichannel serial ports (MCSPs)
- Interrupt Selector
- 32-bit timers
- Power-down logic

Figure 1–1 shows the block diagram for peripherals for the TMS320C62xx devices.

Figure 1–1. TMS320C6x Block Diagram



The direct memory access (DMA) controller transfers data between address ranges in the memory map without intervention by the CPU. The DMA controller allows movement of data to and from internal memory, internal peripherals, or external devices to occur in the background of CPU operation. The DMA controller has four independent programmable channels, allowing four simultaneous contexts for DMA operation. In addition, a fifth (auxiliary) channel allows the DMA controller to service a request from the host port interface (HPI).

The HPI is a parallel port through which a host processor can directly access the CPU's memory space. The host device is the master of the interface, therefore increasing its ease of access by the host. The host and the CPU can exchange information via 'C6201 internal or external memory. In addition, the host has direct access to memory-mapped peripherals.

The external memory interface (EMIF) supports a glueless interface to several external devices including:

- Synchronous burst SRAM (SBSRAM)
- Synchronous DRAM (SDRAM)
- Asynchronous devices, including SRAM, ROM, and FIFOs
- An external shared-memory device

The TMS320C62xx provides a variety of boot configurations for proper device initialization. These configurations determine what actions the 'C62xx performs after device reset to prepare for initialization.

The 'C62xx multichannel serial port (MCSP) is based on the standard serial port interface found on the TMS320C2x, 'C2xx, 'C5x, and 'C54x devices. In addition, the port has the ability to buffer serial samples in memory automatically with the aid of the DMA controller. It also has multichannel capability compatible with the T1, E1, and MVIP networking standards. Like its predecessors, it provides:

- Full duplex communication
- Double buffered data registers which allow a continuous data stream
- Independent framing and clocking for receive and transmit
- Direct interface to industry standard codecs, analog interface chips (AICs), and other serially connected A/D and D/A devices
- Either an external shift clock generation or an internal programmable frequency shift clock

In addition, the MCSP has the following capabilities:

- Direct interface to:
  - T1/E1 framers
  - ST–BUS compliant devices
  - IOM-2 compliant devices
  - AC97 compliant devices
  - IIS compliant devices
  - SPI™ devices
- Multichannel transmit and receive of up to 128 channels
- A wider selection of data sizes including 8, 12, 16, 20, 24, or 32 bits
- µ-Law and A-Law companding
- 8-bit data transfers with LSB or MSB first
- Programmable polarity for both frame synchronization and data clocks
- Highly programmable internal clock and frame generation

The ’C62xx has two 32-bit general-purpose timers that are used to:

- Time events
- Count events
- Generate pulses
- Interrupt the CPU
- Send synchronization events to the DMA controller

The ’C62xx peripheral set produces 16 interrupt sources. The CPU has 12 interrupts available for use. The interrupt selector allows you to choose which 12 of the 16 your system needs to use. The interrupt selector also allows you to effectively change the polarity of external interrupt inputs.

The power-down logic allows reduced clocking to reduce power consumption. Most of the operating power of CMOS logic dissipates during circuit switching from one logic state to another. By preventing some or all of the chip’s logic from switching, significant power savings can be realized without losing any data or operational context.

Chapter 2

Internal Program Access

Describes the TMS320C6201 program memory system. This includes program memory mode and cache modes.

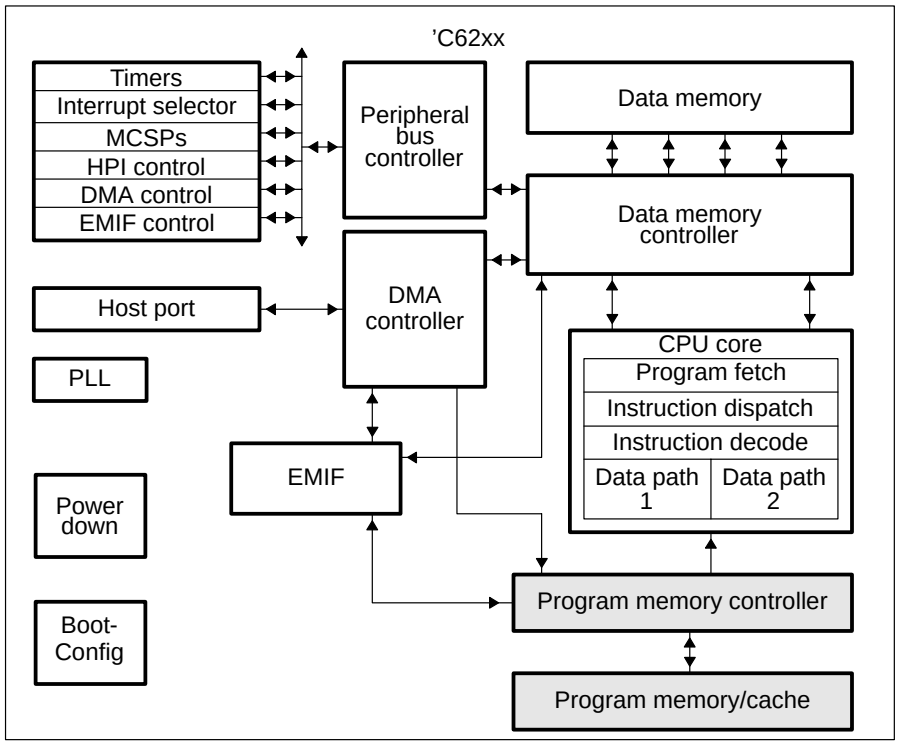
Topic **Page**

2.1 Overview

As shown in Figure 2–1, the Program Memory Controller:

- Performs CPU and DMA requests to internal program memory and necessary arbitration.
- Performs CPU requests to external memory through the external memory interface (EMIF).
- Manages the internal program memory when configured as cache.

Figure 2–1. TMS320C6x Block Diagram



2.2 Internal Program Memory

The internal program memory includes 64K bytes of RAM or equivalently 2K 256-bit fetch packets or 16K 32-bit instructions. The CPU through the program memory controller has a single-cycle throughput 256-bit wide connection to internal program memory.

2.2.1 Internal Program Memory Modes

The internal program memory can be utilized in four modes selected by the Program Cache Control (PCC) field (bits 7–5) in the CPU Control and Status Register as shown in Table 2–1. The modes are:

0. **Mapped:** Depending on the memory map selected, the program memory either is located at either address:

   - 0000 0000h–0000 FFFFh in Map 1.
   - 0140 0000h–0140 FFFFh in Map 0.

   Refer to the Chapter 7 Boot Configuration, Reset, and Memory Map on how to select the memory map. In mapped mode, program fetches to the internal program memory address return the fetch packet at that address. In any cache mode, CPU accesses to this address range returns the fetch packet containing all NOPs. Mapped mode is the default state of the internal program memory at reset.

1. **Enabled:** In cache enabled mode, any initial program fetch of an address causes a cache miss. In a cache miss, the fetch packet is loaded from the external memory interface (EMIF) and stored in the internal cache memory one 32-bit instruction at a time. When all 8 instructions in the fetch packet are received, it is sent to the CPU for execution. During this period the CPU is halted in it entirety. The number of wait-states incurred depends on the type of external memory used, the state of that memory, and any contention for the EMIF with other requests such as the DMA or CPU data access. Any subsequent read from a cached address will cause a cache hit and that fetch packet is sent to the CPU from the internal program size without any wait-states. On the change from program memory mode to cache enabled mode, the program cache is flushed. This mode transition is the only means to flush the cache.

2. **Freeze:** During a cache freeze, the cache retains its current state. A program read to a frozen cache is identical to a read to an enabled cache with the exception that on a cache miss the data read from the external memory interface is not stored in the cache. A subsequent read of the same address will also cause a cache miss and the data will again be

fetched from external memory. Cache freeze can ensure that critical program data is not overwritten in the cache.

3. **Bypass:** When the cache is bypassed, any program read will fetch data from external memory. The data is not stored in the cache memory. Like cache freeze, in cache bypass the cache retains its state. This mode ensures that external program data is being fetched.

Table 2–1. Internal Program Memory Modes

Program Mode	PCC Value	Description
Mapped	000	Memory mapped. Cache Disabled. (Reset default.)
Cache Enable	010	Cache accessed and updated on reads.
Cache Freeze	011	Cache accessed but not updated on reads.
Cache Bypass	100	Cache not accessed or updated on reads.
Reserved	Other	

2.2.2 Cache Architecture

The architecture of the cache is direct mapped. The 64K byte cache contains 2K fetch packets and thus 2K frames. The width of the cache (the frame size) is 256-bits. Each frame in the cache is one fetch packet.

2.2.2.1 Cache Usage of CPU Addresses

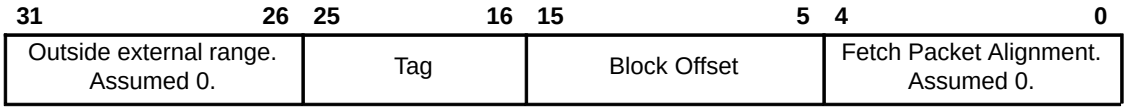
Figure 2–2 shows how the cache uses the fetch packet address from the CPU:

5-bit Fetch Packet Alignment: The 5 LSBs of the address are assumed to be zero because all program fetch requests are aligned on fetch packet boundaries (8 words or 32 bytes).

11-bit Tag Block Offset: Because the cache is direct mapped, any external address maps to only one of the 2K frame. Any two fetch packets that are an even multiple of 64K byte addresses apart map to the same frame. Thus, bits 15:5 of the CPU address create the 11-bit block offset that determines which of the 2K frames any particular fetch packet maps to.

10-bit Tag: The cache assumes an a maximum external address space of 64Mbytes (from 00000000–03FFFFFF). Thus, bits 25:16 correspond to the tag that determines the original location of the fetch packet in external memory space. The cache also has a separate 2Kx11 tag RAM that holds all the tags. This RAM contains the 10-bit tag plus a valid bit which is used in cache flush.

Figure 2–2. Logical Mapping of Cache Address



2.2.2.2 Cache Flush

A dedicated valid bit in the tag RAM indicates whether the contents of that cache frame contains valid data. During a cache flush all these valid bits are cleared to indicate that no cache frames have valid data. Cache flushes only occur at the transition of the internal program memory from mapped mode to cache enabled mode.

2.2.2.3 Frame Replacement

A cache miss is detected when the tag corresponding to the block offset of the fetch packet address requested by the CPU does not correspond to bits 25:16 of the fetch packet address or if the valid bit at that location is clear. If enabled, the cache loads the fetch packet into the corresponding frame, sets the valid bit, sets the tag to bits 25:16 of the requested address, and delivers this fetch packet to the CPU after all 8 instructions are available.

2.3 DMA Access to Program Memory

The DMA can read and write to internal program memory when configured in memory-mapped mode. The CPU always has priority over the DMA for access to internal program memory regardless of the value of the PRI bit for that DMA channel. DMA accesses are postponed until the CPU stops making requests. In a cache mode, a DMA write is ignored by the Program Memory Controller. In a cache mode, read returns an undefined value. For both DMA reads and writes in cache modes, the DMA is signaled that its request has completed. At reset, the program memory system is in mapped mode. This feature allows the DMA to bootload code into the internal program memory. See Chapter 7 Boot Configuration, Reset, Memory Map for more information on bootloading code.

Chapter 3

Internal Data Access

Describes the internal data memory organization, and CPU/DMA data access arbitration.

Topic **Page**

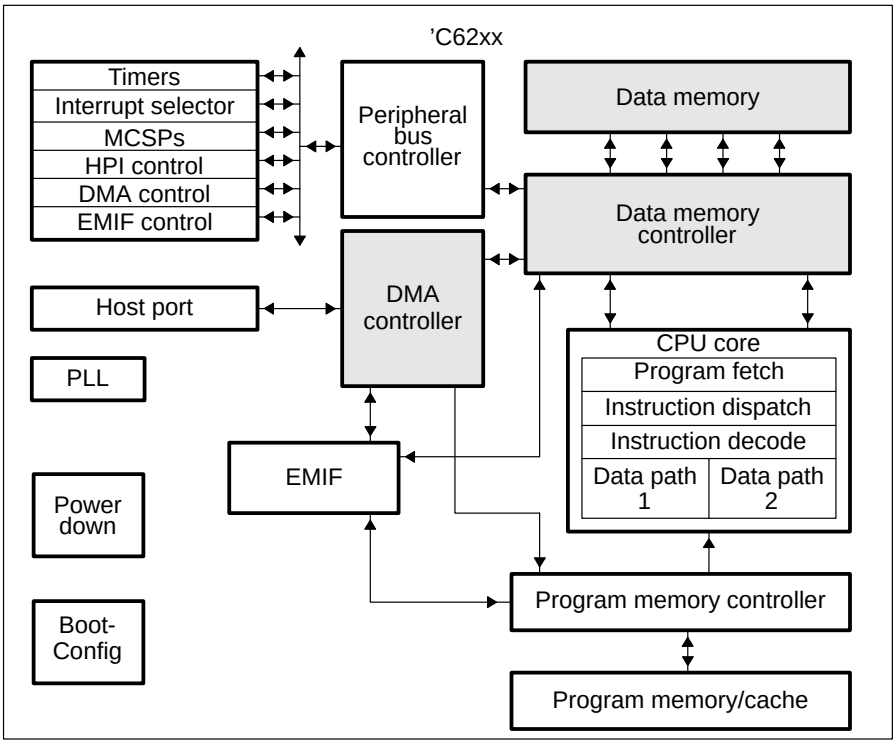
3.1 Overview

As shown inFigure 3–1, the Data Memory Controller connects:

- The CPU and Direct Memory Access Controller (DMA) to internal data memory and performs the necessary arbitration.
- CPU to the external memory interface (EMIF).
- The CPU to the on chip peripherals through the Peripheral Bus Controller.

The peripheral bus controller performs arbitration between the CPU and DMA for the on-chip peripherals.

Figure 3–1. TMS320C6x Block Diagram



3.2 Data Memory Access

The Data Memory Controller services all requests to internal memory as well as all CPU data requests. Figure 3–2 shows the directions of data flow as well as the master (requester) and slave (resource) relationships between the modules:

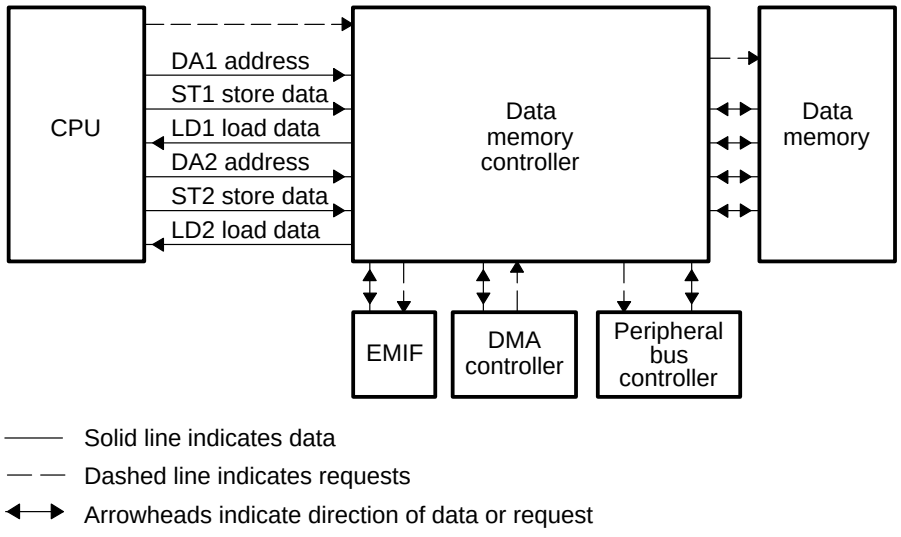
- CPU requests data reads and writes to:

1) internal data memory.
2) on-chip peripherals through the peripheral bus controller.
3) EMIF.

- DMA requests reads and writes to internal data memory.

CPU sends requests to the Data Memory Controller through the two address buses (DA1 and DA2). Store data is transmitted through the CPU data store buses (ST1 and ST2). Load data is received through the CPU data load buses (LD1 and LD2). The CPU data requests are mapped based on address to either the internal data memory, internal peripheral space through the peripheral bus controller, or the external memory interface. The Data Memory Controller also connects the DMA to the internal data memory and performs CPU/DMA arbitration for the on-chip data RAM.

Figure 3–2. Data Memory Controller Interconnect to Other Blocks



3.3 Internal Data Memory Organization

The 64K bytes of internal data RAM located from address 8000 0000h to 8000 FFFFh is organized as four 8K blocks of 16-bit halfwords. These blocks are organized in an interleave on 16-bit data with each block containing successive half-words. This interleave allows the two data ports and DMA to read neighboring 16-bit data elements without a resource conflict.

Table 3–1. Data Memory Organization

	8n+0 Block		8n+2 Block		8n+4 Block		8n+6 Block	
First Address	80000000 80000008	80000001 80000009	80000002 8000000A	80000003 8000000B	80000004 8000000C	80000005 8000000D	80000006 8000000E	80000007 8000000F
	8000FFF0	8000FFF1	8000FFF2	8000FFF3	8000FFF4	8000FFF5	8000FFF6	8000FFF7
Last Address	8000FFF8	8000FFF9 DA1	8000FFFA byte	8000FFFB halfword	8000FFFC word	8000FFFD	8000FFFE	8000FFFF

3.3.1 Data Alignment

Both the CPU and DMA can read and write 8-bit bytes, 16-bit halfwords, and 32-bit words. The data memory controller performs arbitration between the two independently for each 16-bit block basis. The following data alignment restrictions apply:

Words: Words are aligned on even four-byte boundaries (word-boundaries). Words always start at a byte address where the two LSBs are 0. A word access requires two adjacent 16-bit wide blocks.

Halfwords: Halfwords are aligned on even two-byte boundaries (halfword boundaries). Halfwords always start at byte addresses where the LSB is 0. Halfword accesses require the entire 16-bit wide block.

Bytes: There are no alignment restrictions related to byte accesses. Although arbitration is done on 16-bit wide blocks, the blocks still have byte enables to support byte-wide accesses. However, a byte access requires the entire 16-block the byte address maps to.

3.3.2 Dual CPU Accesses to Internal Memory

Interleaved memory organization allows two CPU simultaneous memory accesses. In one CPU cycle, two simultaneous accesses to two different internal memory blocks occur without wait-states. Two simultaneous accesses to the same internal memory block stall the entire CPU pipeline for one CPU clock, providing two accesses in two CPU clocks. These rules apply regardless of whether the accesses are loads or stores. Loads and stores from the same execute packet are seen by the Data Memory Controller during the same CPU cycle. Loads and stores from future or previous CPU cycles do not cause wait-states for the internal data memory accesses in the current cycle. Thus, internal data memory access only causes a wait-state when conflicts occur between instructions in the same fetch packet access the same 16-bit wide block. This condition is called an internal memory conflict. Here, the Data Memory Controller stalls the CPU for one additional CPU clock, serializes the accesses, and performs each access separately. In prioritizing the two accesses, any load occurs before any store access. If both accesses are loads or both accesses are stores, the access from DA1 takes precedence over the access from DA2. Figure 3–3 shows what access conditions cause internal memory conflicts given the when the CPU makes two data accesses (on DA1 and DA2).

Figure 3–3. Conflicting Internal Memory Accesses

	DA1	**Byte**								**Halfword**				**Word**	
DA2	2:0	000	001	010	011	100	101	110	111	000	010	100	110	000	110
Byte	000	■	■							■				■	
	001	■	■							■				■	
	010			■	■						■			■	
	011			■	■						■			■	
	100					■	■					■			■
	101					■	■					■			■
	110							■	■				■		■
	111							■	■				■		■
Halfword	000	■	■							■				■	
	010			■	■						■			■	
	100					■	■					■			■
	110							■	■				■		■
word	000	■	■	■	■					■	■			■	
	110					■	■	■	■			■	■		■

Note: Conflicts shown in shaded areas.

3.3.3 DMA Accesses to Internal Memory

If a DMA access to internal memory does not require the same 16-bit banks used by any CPU accesses, the DMA operation occurs in the background. You can use Figure 3–3 to determine DMA versus CPU conflicts. Assume that one axis represents the DMA access and the other represents the CPU access from one CPU data port. Then, perform this analysis again for the other data port. If both comparisons yield no conflict, then there is no CPU/DMA internal memory conflict. Here, the CPU access and DMA access occur in the background of each other. If either comparison yields a conflict, then there is a CPU/DMA internal memory conflict. In this case, the priority is resolved by the PRI bit of the DMA Channel as described in Chapter 4 of this reference guide. If the DMA Channel is configured as higher priority than the CPU (PRI = 1), any CPU accesses are postponed until the DMA accesses complete and the CPU incurs a one CPU clock wait state. If both CPU ports and the DMA access the same memory block, the number of wait-states increases to two. If the DMA has multiple consecutive requests to the block required by the CPU, the CPU is held off until all DMA accesses to the necessary blocks complete. In contrast, if the CPU is higher priority (PRI = 0), then the DMA access is postponed until the both CPU data ports stop accessing that block. In this configuration a DMA access request never causes a wait-state.

3.3.4 Data Endianness

Two standards for data ordering in byte-addressable microprocessors exist:

- Little endian
- Big endian

The CPU and DMA support a programmable endianness. This endianness is selected by the LENDIAN (Little ENDIAN) pin on the device. LENDIAN = 1 and LENDIAN = 0 selects little endian and big endian, respectively. This selection applies to both the CPU and the DMA. Byte ordering within word and half-word data resident in memory is identical for little endian and big endian data. Table 3–2 shows which bits of a data word in memory are loaded into which bits of a destination register for all possible CPU data loads from big or little endian data. The data in memory is assumed to be the same data that is in the register results from the LDW instruction in the first row. Table 3–3 and Table 3–4 show which bits of a register are stored in which bits of a destination memory word for all possible CPU data stores from big and little endian data. The data in the source register is assumed to be the same data that is in the memory results from the STW instruction in the first row.

Table 3–2. Register Contents After Little Endian or Big Endian Data Loads

Instruction	Address Bits (1:0)	Big Endian Register Result	Little Endian Register Result
LDW	00	BA987654h	BA987654h
LDH	00	FFFFBA98h	00007654h
LDHU	00	0000BA98h	00007654h
LDH	10	00007654h	FFFFBA98h
LDHU	10	00007654h	0000BA98h
LDB	00	FFFFFFBAh	00000054h
LDBU	00	000000BAh	00000054h
LDB	01	FFFFFF98h	00000076h
LDBU	01	00000098h	00000076h
LDB	10	00000076h	FFFFFF98h
LDBU	10	00000076h	00000098h
LDB	11	00000054h	FFFFFFBAh
LDBU	11	00000054h	000000BAh

Note: The contents of the word in data memory at location “xxxx xx00” is BA987654h.

Table 3–3. Memory Contents After Little Endian or Big Endian Data Stores

Instruction	Address Bits (1:0)	Big Endian Memory Result	Little Endian Memory Result
STW	00	BA987654h	BA987654h
STH	00	76541970h	01127654h
STH	10	01127654h	76541970h
STB	00	54121970h	01121954h
STB	01	01541970h	01125470h
STB	10	01125470h	01541970h
STB	11	01121954h	54121970h

Note: The contents of the word in data memory at location “xxxx xx00” before the ST instruction executes is 01121970h. The contents of the source register is BA987654h.

Table 3–4. Memory Contents After Little Endian or Big Endian Data Stores

Access Type	Address Bits (1:0)	Big Endian Register	Little Endian Memory Result
Word	00	XXXXXXXX	XXXXXXXX
Halfword	00	XXXX????	????XXXX
Halfword	10	????XXXX	XXXX????
Byte	00	XX??????	??????XX
Byte	01	??XX????	????XX??
Byte	10	????XX??	??XX????
Byte	11	??????XX	XX??????

Note: X indicates nybbles correctly written, ? indicates nybbles with undefined value after write

3.4 Peripheral Bus

The peripherals described in this reference guide are controlled by CPU and DMA through accesses of control registers. The CPU and DMA access these registers through the peripheral data bus. The DMA directly accesses the Peripheral Bus Controller, whereas CPU accesses it through the Data Memory Controller.

3.4.1 Byte and Halfword Access

The peripheral bus controller converts all peripheral bus accesses to word accesses. However, on read accesses both the CPU and DMA can extract the correct portions of the word to perform byte and halfword accesses properly. However, any side-effects by causing a peripheral control register read will occur regardless of which bytes are read. In contrast, for byte or halfword writes, the values the CPU and DMA only provide correct values in the enabled bytes. The values guaranteed to be correct are shown in Table 3–4. Thus, undefined results will be written to the non-enabled bytes. If the user is not concerned about the values in the disabled bytes, this is acceptable. Otherwise, you should only access the peripheral registers via word accesses.

3.4.2 CPU Wait States

Isolated peripheral bus controller accesses from the CPU causes 4 CPU wait states. These wait-states are inserted to allow pipeline registers to break-up the paths between traversing the on-chip distances between the CPU and peripherals as well as for arbitration time. On consecutive accesses, an access after the first only require 3 CPU wait-states due to the pipelined nature of the Data Memory Controller's interface to the peripheral bus controller.

3.4.3 CPU/DMA Arbitration

As shown in Figure 3–2, the Peripheral Bus Controller performs arbitration between the CPU and DMA for the peripheral bus. Like internal data access, the PRI bits in the DMA determine the priority between the CPU and the DMA. If a conflict occurs between the CPU (via the Data Memory Controller) the lower priority requester is held off until the higher priority requester completes all accesses to the peripheral bus controller. The peripheral bus is arbitrated as a single resource, thus the lower priority resource is blocked from accessing all peripherals, not just the one accessed by the higher priority requester.

Chapter 4

Direct Memory Access (DMA) Controller

Describes the direct memory access channels and registers available for the TMS320C62xx devices.

Topic **Page**

4.1 Overview

The direct memory access (DMA) controller transfers data between regions in the memory map without intervention by the CPU. The DMA allows movement of data to and from internal memory, internal peripherals, or external devices to occur in the background of CPU operation. The DMA has four independent programmable channels allowing four different contexts for DMA operation. Each DMA channel can be independently configured to transfer data elements of different sizes: 8-bit bytes, 16-bit half-words, or 32-bit words. In addition a fifth (auxiliary) channel allows the DMA to service requests from the host port interface(HPI). In discussing DMA operations several terms are important:

- **Read transfer:** The DMA reads the data element from a source location in memory.
- **Write transfer:** The DMA writes the data element that was read during a read transfer to its destination location in memory.
- **Element transfer:** The combined read and write transfer for a single data element.
- **Frame transfer:** Each DMA channel has an independently programmable number of elements per frame. In completing a frame transfer, the DMA moves all elements in a single frame.
- **Block transfer:** Each DMA channel also has an independently programmable number of frames per block. In completing a block transfer, the DMA moves all frames it has been programmed to move.

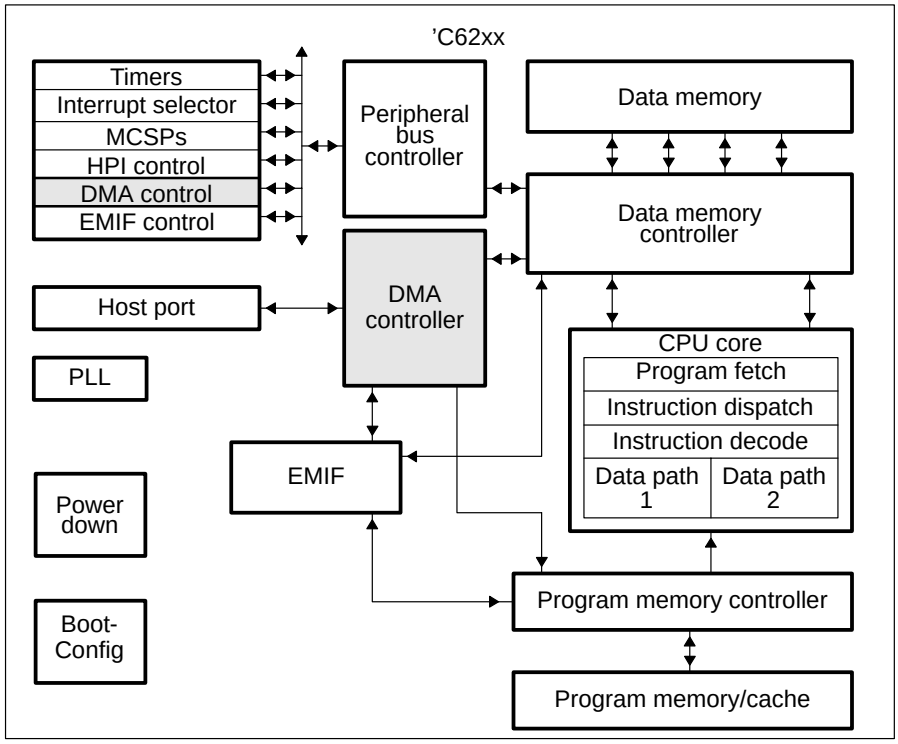
The DMA has the following features:

- **Background operation:** The DMA operates independently of the CPU.
- **High throughput:** Elements can be transferred at the CPU clock rate. See Section 4.11.
- **Four channels:** The DMA can keep track of the contexts of four independent block transfers. See Section 4.2.
- **Auxiliary channel:** This simple channel allows the host port to make requests into the CPU's memory space. This chapter discusses how the auxiliary channel requests are prioritized relative to other channels and the CPU. Detailed explanation of how it is used in conjunction with a peripheral is found in that peripheral's documentation.
- **Split operation:** A single channel maybe used to simultaneously perform both the receive and transmit element transfers from or to a peripheral, ef-

fectively acting like two DMA channels without the additional cost. See Section 4.8.

- **Multiframe transfer:** Each block transfer can consist of multiple frames of a fixed programmable size. See Section 4.5.
- **Programmable priority:** Each channel has independently programmable priorities versus the CPU for each of the memory-mapped resources.
- **Programmable address generation:** Each channel's source and destination address registers can have configurable indexes through memory on each read and write transfer. The address may remain constant, increment, decrement, or be adjusted by a programmable value. The programmable value allows a distinct index for the last transfer in a frame and for the preceding transfers. See subsection 4.7.1.
- **Full-address 32-bit address range:** The DMA can access any region in the memory map:
  - The on-chip data memory.
  - The on-chip program memory when mapped into memory space rather than being utilized as cache.
  - On-chip peripherals.
  - The external memory interface (EMIF).
- **Programmable width transfers:** Each channel can be independently configured to transfer either bytes, 16-bit half-words, or 32-bit words. See subsection 4.7.3.
- **Autoinitialization:** Once a block transfer is complete, a DMA channel may automatically re-initialize itself for the next block transfer. See subsection 4.4.1.
- **Event synchronization:** Each read, write, or frame transfer may be initiated by selected events. See Section 4.6.
- **Interrupt generation:** On completion of each frame transfer or of an entire block transfer as well a on various error conditions, each DMA channel may send an interrupt to the CPU. See Section 4.10.

Figure 4–1. DMA Controller Interconnect to TMS320C6x; Memory Mapped Modules



4.2 DMA Registers

The DMA registers are essential in configuring the operation of the DMA. Table 4–1 and Table 4–2 show how the DMA control registers are mapped into the CPU's memory space. These registers include the DMA global data, count reload, index, and addess registers as well as number of independent control registers for each channel. The DMA global data registers are useful for a variety of functions. (Figure 4–2 and Figure 4–3).

Table 4–1. DMA Control Registers by Address

Hex Byte Address	Name	Section
01840000	DMA channel 0 primary control	4.2.1
01840004	DMA channel 2 primary control	4.2.1
01840008	DMA channel 0 secondary control	4.10
0184000C	DMA channel 2 secondary control	4.10
01840010	DMA channel 0 source address	4.7
01840014	DMA channel 2 source address	4.7
01840018	DMA channel 0 destination address	4.7
0184001C	DMA channel 2 destination address	4.7
01840020	DMA channel 0 transfer counter	4.5
01840024	DMA channel 2 transfer counter	4.5
01840028	DMA global reload register A	4.2
0184002C	DMA global reload register B	4.2
01840030	DMA global index register A	4.2
01840034	DMA global index register B	4.2
01840038	DMA global address register A	4.2
0184003C	DMA global address register B	4.2
01840040	DMA channel 1 primary control	4.2.1
01840044	DMA channel 3 primary control	4.2.1
01840048	DMA channel 1 secondary control	4.10
0184004C	DMA channel 3 secondary control	4.10
01840050	DMA channel 1 source address	4.7
01840054	DMA channel 3 source address	4.7
01840058	DMA channel 1 destination address	4.7
0184005C	DMA channel 3 destination address	4.7
01840060	DMA channel 1 transfer counter	4.5
01840064	DMA channel 3 transfer counter	4.5
01840068	DMA global address register C	4.2
0184006C	DMA global address register D	4.2
01840070	DMA auxiliary control register	4.9.1

Table 4–2. DMA Control Registers by Register Name

Name	Hex Byte Address	Section
DMA channel 0 destination address	01840018	4.7
DMA channel 0 primary control	01840000	4.2.1
DMA channel 0 secondary control	01840008	4.10
DMA channel 0 source address	01840010	4.7
DMA channel 0 transfer counter	01840020	4.5
DMA channel 1 destination address	01840058	4.7
DMA channel 1 primary control	01840040	4.2.1
DMA channel 1 secondary control	01840048	4.10
DMA channel 1 source address	01840050	4.7
DMA channel 1 transfer counter	01840060	4.5
DMA channel 2 destination address	0184001C	4.7
DMA channel 2 primary control	01840004	4.2.1
DMA channel 2 secondary control	0184000C	4.10
DMA channel 2 source address	01840014	4.7
DMA channel 2 transfer counter	01840024	4.5
DMA channel 3 destination address	0184005C	4.7
DMA channel 3 primary control	01840044	4.2.1
DMA channel 3 secondary control	0184004C	4.10
DMA channel 3 source address	01840054	4.7
DMA channel 3 transfer counter	01840064	4.5
DMA auxiliary control register	01840070	4.9.1
DMA global reload register A	01840028	4.2
DMA global reload register B	0184002C	4.2
DMA global index register A	01840030	4.2
DMA global index register B	01840034	4.2
DMA global address A	01840038	4.2
DMA global address B	0184003C	4.2
DMA global address C	01840068	4.2
DMA global address D	0184006C	4.2

Figure 4–2. DMA Global Data Register Diagram

Figure 4–3. DMA Global Data Register Uses

DMA global data register	Source addr reload	Destination addr reload	Count reload	Index	Split address
Count reload A			■		
Count reload B			■		
Index A				■	
Index B				■	
Address A					■
Address B	■	■			■
Address C	■	■			■
Address D	■	■			

4.2.1 DMA Channel Control Registers

The DMA channel primary (Figure 4–4) and secondary control register (Figure 4–5) contain bit-fields that control each individual DMA channel independently. These fields are described in Table 4–3 and Table 4–4.

Figure 4–4. DMA Channel Primary Control Register

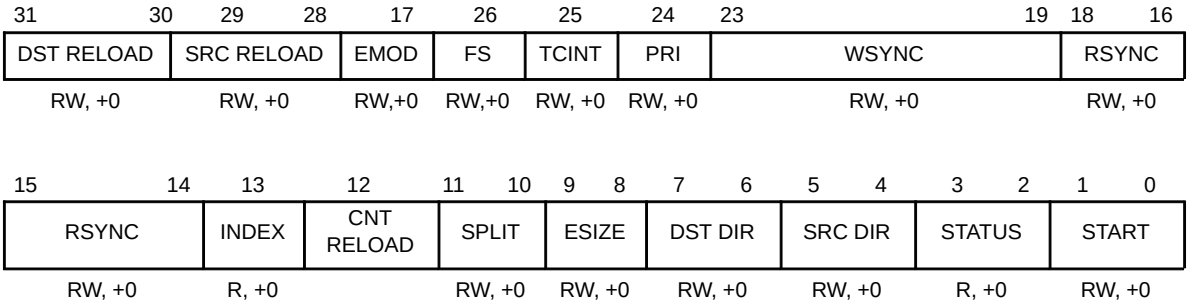


Figure 4–5. DMA Channel Secondary Control Register

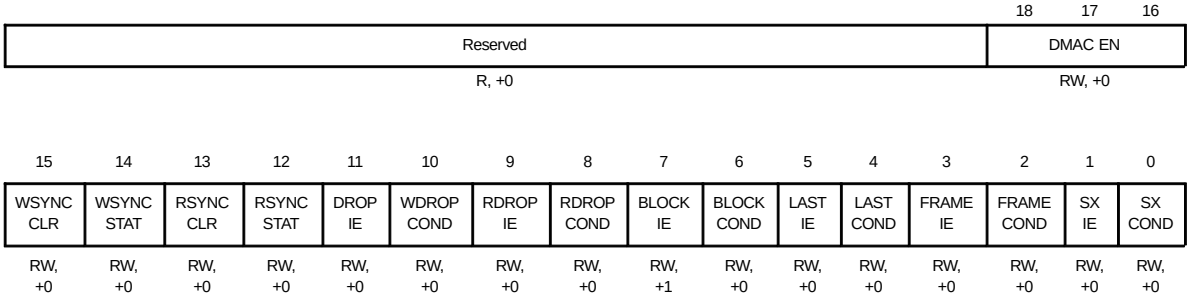


Table 4–3. DMA Channel Secondary Control Register Bitfields

Bitfield	Description	Section
SX COND FRAME COND LAST COND BLOCK COND (R/W)DROP COND	DMA condition. See 4.10 for description. COND = 0, condition not detected COND = 1, condition detected	4.10
SX IE FRAME IE LAST IE BLOCK IE (R/W)DROP IE	DMA condition interrupt enable. See 4.10.1 for description. IE = 0, associated condition doesn't enable DMA channel interrupt IE = 1, associated condition enables DMA channel interrupt	4.10.1
(R/W)SYNC STAT	Read, write synchronization status Write 1 to set associated status. STAT = 0, synchronization not received STAT = 1, synchronization received	4.6.1
DMAC EN	DMAC pin control DMAC EN = 000b, DMAC pin held low DMAC EN = 001b, DMAC pin held high DMAC EN = 010b, DMAC reflects RSYNC STAT DMAC EN = 011b, DMAC reflects WSYNC STAT DMAC EN = 100b, DMAC reflects FRAME COND DMAC EN = 101b, DMAC reflects BLOCK COND DMAC EN = other, reserved	4.12
(R/W)SYNC CLR	Read, write synchronization status clear Read as 0, write 1 to clear associated status.	4.6.1

Table 4–4. DMA Channel Primary Control Register Field Definitions

Bitfield	Description	Section
START	START = 00b, stop START = 01b, start, without auto-initialization START = 10b, pause START = 11b, start with auto-initialization	4.4
STATUS	STATUS = 00b, stopped STATUS = 01b, running, without auto-initialization STATUS = 10b, paused STATUS = 11b, running, with auto-initialization	4.4
SRC DIR, DST DIR	Source/destination address modification after element transfers (SRC/DST) DIR = 00b, no modification. (SRC/DST) DIR = 01b, increment by element size in bytes (SRC/DST) DIR = 10b, decrement by element size in bytes. (SRC/DST) DIR = 11b, adjust using DMA global data register selected by INDEX.	4.7.1
RSYNC, WSYNC	Read transfer/write transfer synchronization. (R/W)SYNC = 00000b, no synchronization. (R/W)SYNC = other sets synchronization event	4.6
FS	Frame synchronization FS = 0, disable, FS = 1, RSYNC event used to synchronize entire frame.	4.6
TCINT	Transfer controller interrupt. TCINT = 0 interrupt disabled TCINT = 1 interrupt enabled	4.10
ESIZE	Element size ESIZE = 00b, 32-bit ESIZE = 01b, 16-bit ESIZE = 10b, 8-bit ESIZE = 11b, reserved	4.7.3
PRI	Priority mode: DMA v. CPU PRI = 0, CPU priority PRI = 1, DMA priority	4.9
SPLIT	Split channel mode. SPLIT = 00b disabled SPLIT = 01b, enabled, use DMA global address register A as split address. SPLIT = 10b, enabled, use DMA global address register B as split address. SPLIT = 11b, enabled, use DMA global address register C as split address.	4.8

Table 4–4. DMA Channel Primary Control Register Field Definitions (Continued)

Bitfield	Description	Section
CNT RELOAD	DMA channel transfer counter reload for auto-initialization and multi-frame transfers CNT RELOAD = 0, Reload with DMA global count reload register 0 CNT RELOAD = 1, Reload with DMA global count reload register 1	4.4.1.1
INDEX	Selects the DMA global data register to use as a programmable index. INDEX = 0, use DMA global index register 2 INDEX = 1, use DMA global index register 3	4.7.2
EMOD	Emulation mode EMOD = 0, DMA channel keeps running during an emulation halt EMOD = 1, DMA channel paused during an emulation halt	4.13
SRC RELOAD, DST RELOAD	DMA channel source/destination address reload for auto-initialization SRC/DST RELOAD = 00b, do not reload during auto-initialization. SRC/DST RELOAD = 01b, use DMA global address register B as reload. SRC/DST RELOAD = 10b, use DMA global address register C as reload. SRC/DST RELOAD = 11b, use DMA global address register D as reload.	4.4.1.1

4.3 Memory Map

The DMA assumes the superset of the device memory map shown in Chapter 7, *Boot Configuration, Reset, and Memory Map*. Requests are sent to one of four resources:

1) External memory interface
2) Internal program memory
3) Internal peripheral bus
4) Internal data memory

The location of the source and destination are computed at the beginning for a block transfer. Thus, the source address is assumed to point to one of these four spaces throughout a block transfer. This constraint also applies to the destination address.

4.4 Initiating Block Transfer

Each DMA channel may be started independently. This may be done either manually through direct CPU access or through auto-initialization. In addition, each DMA channel may be stopped or paused independently through direct CPU access.

Manual Start Operation: To start DMA operation for a particular channel, once all other DMA control registers are written to their desired values, the DMA channel control register should be written to its desired value with START = 01b. Writing this value to a DMA channel that has already been started has no effect.

Pause Operation: Once started, a DMA channel may then be paused by writing START = 10b. When paused, the DMA channel completes any write transfers whose read transfer requests have completed. Also, if the DMA channel has all of the necessary read synchronizations, one more element additional element transfer will be allowed to complete. Once paused, the value on STATUS is 10b.

Stop Operation: The DMA may also be stopped by writing START = 00b. In this case, the DMA channel stops immediately and discards any data held internally from completed read transfers. The actual status of a DMA channel may be observed by reading the START field in the DMA channel control register. Once a DMA transfer is complete, unless auto-initialization is enabled, the DMA channel returns to the stopped state and STATUS = 00b.

4.4.1 Autoinitialization

The DMA can automatically reinitialize itself after completion of a block transfer. Some of the DMA control registers can be pre-loaded for the next block transfer through reload registers. Selected DMA global data registers act as reload registers. Using this capability some of the parameters of the DMA channel can be set well in advance of the next block transfer. Auto-initialization allows:

- **Continuous Operation:** Continuous operation allows the CPU a long slack time during which it can reconfigure the DMA for a subsequent transfer. Normally, the CPU would have to reinitialize the DMA immediately after completion of the last write transfer in the current block transfer and before the first read synchronization for the next block transfer. In general, with the reload registers, it can reinitialize these values for the next block transfer anytime after the current block transfer begins.
- **Repetitive Operation:** As a special case of continuous operation, once a block transfer completes, the DMA repeats the previous block transfer. In this case, the CPU does not pre-load the reload registers with new values for each block transfer. Instead, the CPU only loads the registers before the first block transfer.

Enabling Auto-Initialization: By writing START = 11b, in the DMA channel control register, auto-initialization is enabled. In this case, after completion of a block transfer, the DMA channel is restarted and the selected DMA channel registers are reloaded. If restarting after a pause, this START must be re-written as 01b for auto-initialization to be enabled.

4.4.1.1 DMA Channel Reload Registers

For auto-initialization, the successive block transfers are assumed to be similar. Thus, the reload values are only selectable for those registers that are modified during a block transfer: the transfer counter and address registers. Thus, the DMA channel transfer counter as well as the DMA channel source and destination address registers have associated reload registers, as selected by the associated RELOAD fields in the DMA channel primary control register (Figure 4–4). The reload registers are stored in particular DMA global address registers.

Note that it is possible to not reload the source or destination address register in auto-initialization mode. This capability allows you to have a register maintain its value that did not change during block transfer. Thus, you do not have to dedicate a DMA global data register to a value that was static during block transfer. A single channel may use the same value for multiple channel registers. For example, in split mode, the source and destination address maybe the same. Similarly, multiple channels may use the same reload values. For example, two channels may have the same transfer count reload value.

Upon completion of a block transfer, these registers are reloaded with the associated reload register. Note that in the case of the DMA channel transfer counter register, reload occurs after the end of each frame transfer, not just after the end of the entire block transfer. The reload value for the DMA channel transfer counter is necessary whenever multi-frame transfers are configured, not just when auto-initialization is enabled.

As discussed in Section 4.11, the DMA may allow read transfers to get ahead of write transfers and provide the necessary buffering to facilitate this capability. To support this, the reload which is necessary at the end of blocks and frames occurs independently for the read (source) and write (destination) portions of the DMA channel. Similarly, in the case of split channel operation described in Section 4.8, the source and destination address are independently reloaded when the associated transmit or receive element transfers are completed.

The DMA channel transfer counter reload can only be rewritten by the user after the next to last frame in the current block transfer completes. Otherwise, the new reload values would affect subsequent frame boundaries in the current block transfer. However, if the frame size is the same for the current and next block transfers, this restriction is not relevant. See Section 4.5 for more explanation of the DMA channel transfer counter.

4.5 Transfer Counting

The DMA channel transfer counter (Figure 4–6) contains bitfields that represent the number of frames and the number of elements per frame to be transferred. Figure 4–7 shows the DMA global count reload register.

FRAME COUNT: This 16-bit unsigned value sets the total number of frames in the block transfer. The maximum number of frames per block transfer is 65535. This counter is decremented upon the completion of the last read transfer in a frame transfer. Once the last frame is transferred, the entire counter is reloaded with the DMA global data register selected by the CNT RELOAD field in the DMA channel primary control register (See Section 4.4.1.1). Also note that initial values of 0 and 1 to FRAME COUNT have the same effect of transferring a single frame.

ELEMENT COUNT: This 16-bit unsigned value sets the number of elements per frame. This counter is decremented after the read transfer of each element. The maximum number of elements per frame transfer is 65535. Once the last element in each frame, is reached, ELEMENT COUNT is reloaded with the 16 LSBs of the DMA global count reload register selected by the CNT RELOAD field in the DMA channel primary control register. This reloading is unaffected by auto-initialization mode. Before block transfer begins, the counter and selected DMA global data register must be loaded with the same 16 LSBs to assure that the first and remaining frames have the same number of elements per frame. In any multi-frame transfer, a reload value must always be specified, not just when auto-initialization is enabled. If the element count is initialized as 0, operation is undefined.

Figure 4–6. DMA Channel Transfer Counter

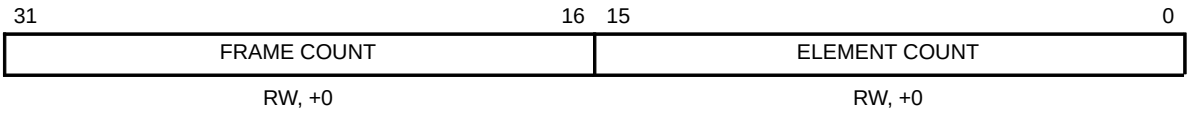
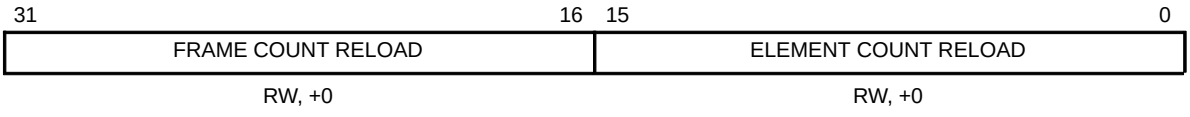


Figure 4–7. DMA Global Count Reload Register Used As Transfer Counter Reload



4.6 Synchronization: Triggering DMA Transfers

Synchronization allows DMA transfers to be triggered by events such as interrupts from internal peripherals or external pins. Three types of synchronization may be enabled for each channel:

1) **Read synchronization:** Each read transfer waits for the selected event to occur before proceeding.

2) **Write synchronization:** Each write transfer waits for the selected event to occur before proceeding.

3) **Frame synchronization:** Each frame transfer waits for the selected event to occur before proceeding.

Selection of Synchronization Events: The events are selected by the RSYNC and WSYNC fields in the DMA channel primary control register. If FS = 1 in this register, then the event selected by RSYNC enables an entire frame. Up to 31 events are available. If the value of these fields is set to 00000b then no synchronization is necessary. In this case, the read, write, or frame transfers occur as soon as the resource is available to that channel. The association between values in these fields to events is shown in Table 4–5. Note that this is very similar to the fields in the interrupt selector. See section 10.5, *Configuring the Interrupt Selector*. One difference is that the MCSP generates separate interrupts and DMA synchronization events. The only other difference is the location of DSPINT in the encoding.

Table 4–5. Synchronization Events

Event Number (Binary)	Event Acronym	Event Description
00000	None	No synchronization
00001	TINT0	Timer 0 interrupt
00010	TINT1	Timer 1 interrupt
00011	SD_INT	EMIF SDRAM timer interrupt
00100	EXT_INT4	External interrupt pin 4
00101	EXT_INT5	External interrupt pin 5
00110	EXT_INT6	External interrupt pin 6
00111	EXT_INT7	External interrupt pin 7
01000	DMA_INT0	DMA channel 0 interrupt
01001	DMA_INT1	DMA channel 1 interrupt

Table 4–5. Synchronization Events (Continued)

Event Number (Binary)	Event Acronym	Event Description
01010	DMA_INT2	DMA channel 2 interrupt
01011	DMA_INT3	DMA channel 3 interrupt
01100	XEVT0	MCSP 0 transmit event
01101	REVT0	MCSP 0 receive event
01110	XEVT1	MCSP 1 transmit event
01111	REVT1	MCSP 1 receive event
10000	DSPINT	Host port host to DSP interrupt
Other	Reserved	

4.6.1 Latching of DMA Channel Event Flags

The DMA channel secondary control register (Table 4–3) contains STAT and CLR fields for read and write synchronization events.

Latching of DMA Synchronization Events: An inactive to active transition of the selected event is latched by each DMA channel. The occurrence of this transition causes the associated STAT field to be set in the DMA channel secondary control register. Note that if no synchronization is selected the STAT bit is always read as 1. Also, note that a single event can trigger multiple actions.

User Clearing and Setting of Events: By clearing pending events before starting a block transfer you can force the DMA channel to wait for the next event. Conversely, by setting events before starting a block transfer you can force the synchronization events necessary for the first element transfer. You may clear or set events (and thus the related STAT bit) by writing 1 to the a corresponding CLR or STAT field, respectively. Note that writing a zero to either of these bits has no effect. Also, the CLR bits are always read as 0 and have no associated storage. Separate bits for setting or clearing are provided to allow clearing of some bits without setting others and vice-versa. Note that user manipulation of events has priority over any simultaneous automated setting or clearing of events.

4.6.2 Automated Event Clearing

The latched STAT for each synchronizing event is only cleared when any action associated with that event completes. Events are cleared as quickly as possible to reduce the minimum time between synchronizing events. This capability effectively increases the throughput at which events can be recognized. This is described in detail for each type of synchronization below:

- **Clearing read synchronization condition:** The latched condition for read synchronization is cleared when the DMA completes the request for the associated read transfer.
- **Clearing write synchronization condition:** The latched condition for write synchronization is cleared when the DMA completes the request for the associated write transfer.
- **Clearing frame synchronization condition:** Frame synchronization clears the RSYNC STAT field when the DMA completes the request for the first read transfer in the new frame.

4.7 Address Generation

For each channel, the DMA performs address computation for each read transfer and write transfer. The DMA allows creation of a variety of data structures. For example, the DMA can traverse an array incrementing through every n^{th} element. Also, it can be programmed to effectively treat the various elements in a frame as coming from separate sources and group each source's data together.

The DMA channel source address and destination address registers (Figure 4–8 and Figure 4–9) hold the addresses for the next read transfer and write transfer, respectively.

Figure 4–8. DMA Channel Source Address Register

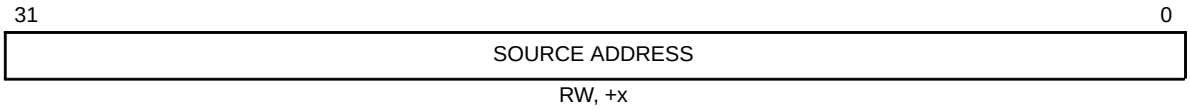
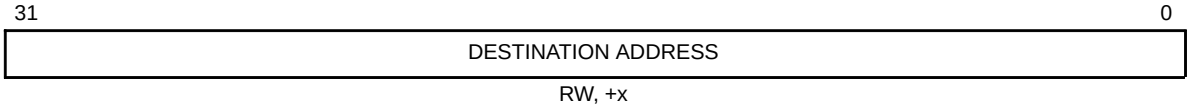


Figure 4–9. DMA Channel Destination Address Register



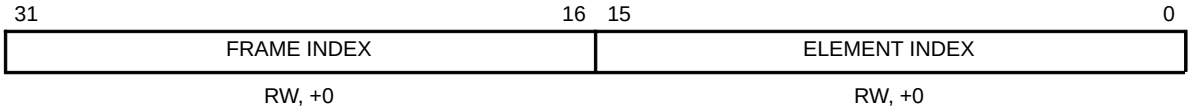
4.7.1 Basic Address Adjustment

As shown in Figure 4–5, the SRC DIR and DST DIR fields can set the index to increment, to decrement, or to not effect the DMA channel source and destination address registers, respectively. By default, these values are set to 00b to disable any incrementing or decrementing. If incrementing or decrementing is enabled, then address adjustment amount is by size of the element in bytes. For example, if the source address is set to increment and 16-bit half-words are being transferred, then the address is incremented by 2 after each read transfer.

4.7.2 Address Adjustment With the DMA Channel Index Registers

As shown in Figure 4–10, the SRC DIR and DST DIR fields can independently allow you to select a particular DMA global index register to determine the address adjustment. The particular DMA global index register is selected via the INDEX field in the DMA channel primary control register. Unlike basic address adjustment, this mode allows different adjustment amount depending on whether or not the element transfer is the last in the current frame. The normal adjustment value (ELEMENT INDEX) is contained in the 16 LSBs of the selected DMA global data register. The adjustment value (FRAME INDEX) for the end of the frame, is determined by the 16 MSBs of the selected DMA global data register. Both of these fields contain signed 16-bit values. Thus, the index amounts can range from –32768 to 32767.

Figure 4–10. DMA Global Index Register



These fields affect address adjustment as follows.

- **ELEMENT INDEX:** For element transfers, except the last one in a frame, ELEMENT INDEX determines the amount to be added to the DMA channel source for the destination address register as selected by the SRC DIR or DST DIR field after each read or write transfer, respectively.
- **FRAME INDEX:** If the read or write transfer is the last in a frame, FRAME INDEX (and not the ELEMENT INDEX) field is used for address adjustment. This adjustment occurs in both single frame and multi-frame transfers.

4.7.3 Element Size, Alignment, and Endianness

Using the ESIZE field in the DMA channel control register, the user may configure the DMA to transfer 8-bit bytes, 16-bit halfwords, or 32-bit words on each transfer. The following registers and bitfields must be loaded with properly aligned values:

- DMA channel source and destination address registers and any associated reload registers.
- ELEMENT INDEX
- FRAME INDEX

In the case of word transfers, these registers must contain values that are multiples of 4, thus aligned on a word address. In the case of half-word transfers they must be multiples of 2, thus aligned on a half-word address. If unaligned values are loaded, operation is undefined. There is no alignment restriction for byte transfers. All accesses to program memory must be 32-bits in width. Also, you must be aware of the endianness when trying to access a particular 8-bit or 16-bit field within a 32-bit register. For example, in little endian, an address ending in 00b select the LSbyte whereas 11b selects the LSbyte in big endian.

4.7.4 Example: Using Frame Index to Reload Addresses

In an auto-initialized, single frame block transfer the FRAME index can be used in place of a reload register to re-compute the next address. In the following example, a single frame transfer moves 10 bytes from a static external address to alternating locations (skip one byte):

- SRC DIR = 00b, static source address.
- DST DIR = 11b, programmable index value
- ELEMENT INDEX = 10b, 2 byte destination stride
- FRAME INDEX = 9x2 = 18 = 10010b, correct by –18 byte locations to restart destination at same place.

4.7.5 Example: Transferring a Large Single Block

The ELEMENT COUNT with the FRAME COUNT can be used in conjunction to effectively allow single frame block transfers of greater than 65535 in size. Here, the product of the element count and frame count can form a larger effective element count. The following must be performed:

- If the address is set to be adjusted using a programmable value (DIR = 11b), the FRAME INDEX must equal the ELEMENT INDEX if the address adjustment is determined by a DMA global index register. This applies to both source and destination addresses. If the address is not set to be adjusted by a programmable value, this constraint does not apply because by default the same address adjustment occurs at element and frame boundaries.
- Frame synchronization must be disabled (FS = 0 in the DMA channel primary control register). This prevents requirements for synchronization in the middle of the large block.
- The number of elements in the first frame is Ei. The number of elements in successive frames is ((F–1) x Er). The effective element count will be ((F–1) x Er) + Ei.

  Where:

  F = The initial value of the FRAME COUNT
  Er = ELEMENT COUNT Reload value
  Ei = initial value of the ELEMENT COUNT

  Thus, to transfer 128K + 1 elements, one could set F = 5, Er = 32K, and Ei = 1.

4.7.6 Example: Sorting

The following procedure is used to have transfers located in memory by ordinal location within a frame (i.e. the first transfer of the first frame followed by the first transfer of the second frame):

- ELEMENT INDEX should be set to: F x S.
- FRAME INDEX be set to: –(((E–1)x F)–1)x S

  Where:

  E = the initial value of ELEMENT COUNT (the number of elements per frame) as well as the ELEMENT COUNT RELOAD.

  F = the initial value of FRAME COUNT (the total number of frames).

  S = the element size in bytes.

Consider a transfer with three frames (F = 3) of four half-word elements each (E = 4, S = 2). This corresponds to ELEMENT INDEX = 3x2 = 6 and FRAME INDEX = –(((4–1)x3) –1)x2 = –16. Assume that the source address is not modified and the destination increments starting at 0x80000000. Table 4–6 and Table 4–7 show how this sorting works for this example.

Table 4–6. Sorting Example in Order of DMA Transfers

Frame	Element	Address	Post Adjustment
0	0	0x80000000	+6
0	1	0x80000006	+6
0	2	0x8000000C	+6
0	3	0x80000012	–16
1	0	0x80000002	+6
1	1	0x80000008	+6
1	2	0x8000000E	+6
1	3	0x80000014	–16
2	0	0x80000004	+6
2	1	0x8000000A	+6
2	2	0x80000010	+6
2	3	0x80000016	–16

Table 4–7. Sorting Grouping Ordered By Address

Frame	Element	Address
0	0	0x80000000
1	0	0x80000002
2	0	0x80000004
0	1	0x80000006
1	1	0x80000008
2	1	0x8000000A
0	2	0x8000000C
1	2	0x8000000E
2	2	0x80000010
0	3	0x80000012
1	3	0x80000014
2	3	0x80000016

4.8 Split Channel Operation

Split channel operation allows a single DMA channel to provide the capability of two channels to service both the input (receive) and output (transmit) streams from an external or internal peripheral with a fixed address.

4.8.1 Split Address Generation

The DMA global address register selected by the SPLIT field in the DMA primary control register determines the address of the peripheral that is to be access for split transfer:

- **Split Source Address:** This address is the source for the input stream to the ’C6x. The selected DMA global address register contains this split source address.
- **Split Destination Dddress:**

- Receive element transfer
  1) **Receive read transfer:** Data is read from the split source address. This event is synchronized as indicated by the RSYNC field.
  2) **Receive write transfer:** Data from the receive read transfer is written to the destination address. The destination address is then adjusted as configured. This event is not synchronized.

Note, since only a single element count and frame count exists per channel, the element count and the frame count are the same for both the received and the transmitted data. For split operation to work properly, both the RSYNC and WSYNC fields must be set to synchronization events. Also, frame synchronization must be disabled in split mode.

For all transfers the above sequence is maintained. However, the transmit transfers do not have to wait for all previous receive element transfers to complete before proceeding. Therefore, it is possible for the transmit stream to get ahead of the receive stream. The DMA channel transfer counter decrements (or reinitialize) after the associated transmit transfer completes. However, re-initialization of the source address register occurs after all transmit element transfers complete. This configuration works as long as transmit transfers do not get eight or more transfers ahead of the receive transfers. If the transmit transfers do get ahead of the receive transfers, transmit element transfers will be stopped, possibly causing missing of synchronization events. For cases where receive or transmit element transfers are within seven or less transfers of the other, the DMA channel maintains this information as internal status.

4.9 Resource Arbitration and Priority Configuration

Priority decides which of competing requesters have control of a resource with multiple requests. The requesters include:

- The DMA channels
- The CPU's program and data accesses.

The resources include:

- Internal data memory including each interleave of internal data memory.
- The internal peripheral registers which are accessed through the peripheral bus.
- Internal program memory.
- The external memory interface (EMIF).

Two aspects of priority are programmable:

DMA versus CPU priority: Each DMA channel may independently be configured in high priority mode by setting the PRI bit in the associated DMA channel control register. The AUXPRI field in the DMA auxiliary control register allows the same feature for the auxiliary channel. When in high priority mode, the associated channel's requests are sent to the appropriate resource with a signal indicating the high priority status. By default all these fields are 0, disabling the high priority mode. Each resource can use this signal in its own priority scheme for resolving conflicts. Refer to the documentation for the particular resource for how it utilizes this signal.

Priority between DMA channels: The DMA has a fixed priority scheme with channel 0 having highest priority and channel 3 having lowest priority. The auxiliary channel may be given a priority anywhere within this hierarchy.

4.9.1 DMA Global Control Register and Priority Between Channels

The fields in the DMA auxiliary control registers affect all the auxiliary channels (Figure 4–12 and Table 4–8). The fields in this register will be described in the following subsections.

Figure 4–12. DMA Channel Global Control Register

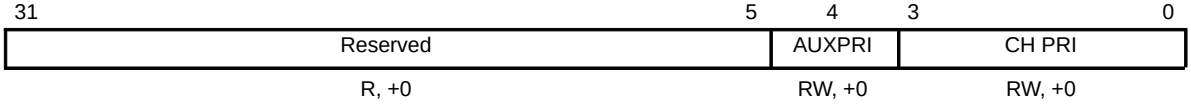


Table 4–8. DMA Auxiliary Control Register

Bitfield	Description
CH PRI	DMA channel priority CH PRI = 0000b, fixed channel priority mode auxiliary channel 1st highest priority CH PRI = 0001b, fixed channel priority mode auxiliary channel 2nd highest priority CH PRI = 0010b, fixed channel priority mode auxiliary channel 3rd highest priority CH PRI = 0011b, fixed channel priority mode auxiliary channel 4th highest priority CH PRI = 0100b, fixed channel priority mode auxiliary channel 5th highest priority CH PRI = other, reserved
AUXPRI	Auxiliary channel priority mode AUXPRI = 0, CPU priority AUXPRI = 1, DMA priority

The priority between DMA channels determines which DMA channel will perform a read or write transfer first, given that two or more channels are ready to perform transfers.

The priority of the auxiliary channel is configurable by programming the CH PRI field in the DMA global control register. By default, CH PRI contains the value 0000b at reset. This value sets the auxiliary channel as highest priority, followed by channel 0, followed by channel 1, followed by channel 2, with channel 3 having lowest priority.

Arbitration between channels occurs independently for read and write transfers every CPU clock cycle. Any channel that is in the process of waiting for synchronization of any kind may lose control of the DMA to a lower priority channel. Once that synchronization is received, that channel may regain control of the DMA from a lower priority channel. This rule is applied independently to the transmit and receive portions of a split mode transfer. The transmit portion has higher priority than the receive portion.

If multiple DMA channels and the CPU are contending for the same resource, the channeler CPU with the highest priority occurs first. Then, arbitration between the highest priority DMA channel and the CPU occurs. Normally, if a channel is lower priority than the CPU, all lower priority channels should also be lower priority than the CPU. Similarly, if a channel has a higher priority than the CPU, all higher priority channels should also be higher priority than the CPU. This contention of the DMA versus CPU arbitration is decided by each resource. Refer to a specific resource's documentation for a full explanation. Note that a channel's PRI field should only be modified when that channel is paused or stopped.

4.9.2 Switching Channels

A higher priority channel will gain control of the DMA from a lower priority channel once it has received the necessary read synchronization. In switching channels, the current channel allows all data from requested reads to complete. The DMA determines which higher priority channel will gain control of the DMA controller read operation. That channel then starts its read operation. Simultaneously, write transfers are allowed to complete from the previous channel.

4.10 DMA Channel Condition Determination

Several condition statuses are available to inform the user of significant events or potential problems in DMA channel operation. These statuses reside in the DMA channel secondary control register COND bit fields.

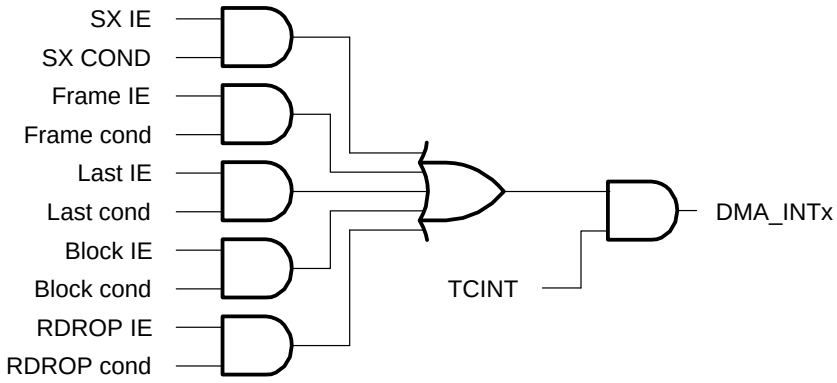
This register also provides the means to enable such events to trigger the DMA to CPU interrupt for a channel through its corresponding interrupt enable (IE) bitfields. If a COND bit and its corresponding IE bit are set, that condition is enabled to contribute to the status of the interrupt signal the from the associated DMA channel to the CPU. If the TCINT bit in the DMA channel *x* control register is set, the logical OR of all enabled conditions forms the DMA_INT*x* signal. Otherwise, the DMA_INT*x* remains inactive. This logic is shown in Figure 4–13. If selected by the interrupt selector, a low to high on that DMA_INT will cause an interrupt condition to be latched by the CPU.

The SX COND, WDROP, and RDROP bits in the DMA channel secondary control register are treated as warning conditions. If these conditions are enabled and active, then they move the DMA channel from the running to the pause state, regardless of the value of the TCINT bit.

If a COND bit's associated IE bit is set, that COND bit may only be cleared by a user write of 0. Otherwise, that COND bit may be automatically cleared. A user write of 1 to a COND bit has no effect. Thus, you cannot manually force one of the conditions.

Most values in this register are cleared at reset. The one exception is the interrupt enable for the block transfer complete event (BLOCK IE), which is set at reset. Thus, by default, the block transfer complete condition is the only condition that could contribute to the CPU interrupt. Other conditions can be enabled by setting the associated IE bit.

Figure 4–13. Generation of DMA Interrupt for Channel x From Conditions



4.10.1 Definition of Channel Conditions

Table 4–9 describes each of the conditions in the DMA channel secondary control register.

Depending on the system application, these conditions may represent errors. The last frame condition can be used to change the reload register values for autoinitialization. The frame index and element count reload are used every frame. Thus, you must wait until all but the last frame transfer in a block transfer completes to change these values. Otherwise, the current block transfer will be affected.

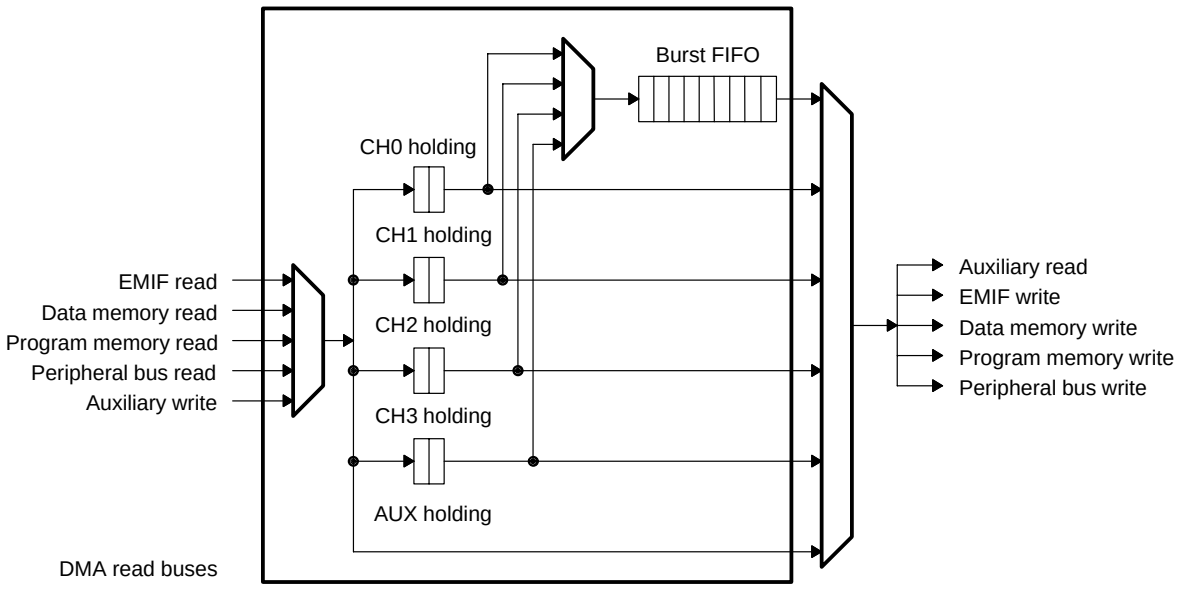
Table 4–9. DMA Channel Condition Descriptions

			COND Cleared By	
Bitfield	**Event**	**Occurs if…**	**If IE Enabled**	**Otherwise**
SX	Split transmit overrun receive	The split operation is enabled and transmit element transfers get seven or more element transfers a ahead of receive element transfers.	A user write of 0 to COND	
FRAME	Frame complete	After the last write transfer in each frame is written to memory.	A user write of 0 to COND.	Two CPU clocks later.
LAST	Last frame	After all counter adjustments for the next to last frame in a block transfer complete.	A user write of 0 to COND.	Two CPU clocks later.
WDROP RDROP	Dropped read/write synchronization	A subsequent synchronization event occurs before the last one is cleared.	A user write of 0 to COND.	
BLOCK	Block transfer complete	After the last write transfer in a block transfer is written to memory.	A user write of 0 to COND.	Two CPU clocks later.

4.11 Structure

Figure 4–14 shows the internal data movement paths of the DMA controller including data buses and internal holding registers.

Figure 4–14. DMA Controller Data Bus Block Diagram



4.11.1 Read and Write Buses

Each DMA channel can independently select one of four sources and destinations:

1) EMIF
2) Internal program memory
3) Internal data memory
4) Internal peripheral bus

Thus, read and write buses from each source interface to the DMA controller.

The auxiliary channel also has read and write buses. However, since the auxiliary channel provides address generation for the DMA, the naming convention of its buses differ. For example, data writes from the auxiliary channel through the DMA are performed through the auxiliary write bus. Similarly, data reads from the auxiliary channel through the DMA are performed through the auxiliary read bus.

4.11.2 DMA FIFO

A 9-deep DMA FIFO holding path is provided to facilitate bursting to high performance memories such as internal program and data memory as well as external synchronous DRAM (SDRAM) or synchronous burst SRAM (SBSRAM). When combined with a channel's holding registers this path effectively becomes an 11-deep FIFO. Only one channel control the FIFO at any given time. For a channel to gain control of the FIFO, the following conditions must all apply:

- The channel does not have read or write synchronization enabled. Since split mode requires read and write synchronization, the FIFO is not used by a channel in split mode. If only frame synchronization is enabled, then the FIFO may still be used by that channel.
- The channel is running.
- The FIFO is void of data from any other channel.
- The channel is the highest priority channel of those that meet the above three conditions.

The third restriction minimizes "head-of-line" blocking. Head-of-line blocking occurs when a DMA request of higher priority waits for a series of lower priority requests to come in before issuing its first request. If a higher priority channel requests control of the DMA controller from a lower priority channel, only the last request of the previous channel has to complete. After that, the higher priority channel completes its requests through its holding registers. The holding registers do not allow as high of a throughput through the DMA controller. In the gaps, the lower priority channel begins no more read transfers but is allowed to flush the FIFO by completing its write transfers. As the higher priority channel is not yet in control of the FIFO, there will be gaps in its access where the lower priority channel may drain its transfer from the FIFO. Once the FIFO is clear, if the higher priority channel has not stopped, it gains control of the FIFO.

The DMA FIFO has two purposes:

- Increasing performance
- Decreasing arbitration latency

For increased performance The FIFO allows read transfers to get ahead of write transfers. This feature minimizes penalties for variations in available transfer bandwidth at either end of the element transfer. Thus, the DMA can capitalize on separate windows of opportunity at the read and write portion of an element transfer. If the requesting DMA channel is using the FIFO, the resources are capable of sustaining read or write accesses at the CPU clock cycle rate. However, there may be some latency in performing the first access. The handshaking between

a resource and the DMA controller controls the rate of consecutive requests and the latency of received read transfer data.

Decreased arbitration latency versus the CPU: The other function of the DMA FIFO is capturing read data from any pending requests for a particular resource. For example, consider the situation where the DMA is reading data from pipelined external memory such as SDRAM or SBSRAM into internal data memory. Assume the CPU is given higher priority over the DMA channel making requests, and that it makes a competing program fetch request to the EMIF. Assume that simultaneously, the CPU is accessing all banks of internal memory, blocking out the DMA. In this case, the FIFO allows the pending DMA accesses to complete and the program fetch to proceed. Due to the pipelined request structure of the DMA, at any one point in time the DMA may have pending read transfer requests whose data has not yet arrived. Once enough requests are outstanding, the DMA stops making further read transfer requests.

4.11.3 Internal Holding Registers

Each channel has dedicated internal holding registers. If a DMA channel is transferring data through its holding registers rather than the internal FIFO, read transfers are issued consecutively. Once a read transfer request has been initiated, no subsequent read transfers are started until the read data has arrived within the holding register. Depending on whether the DMA controller is in split mode or not, additional restrictions can apply:

Split mode: The two registers serve as separate transmit and receive data stream holding registers for split mode. For both the transmit and receive read transfer, no subsequent read transfer request is issued until the associated write transfer request completes.

Non-split mode: Once the data arrives a subsequent read transfer may be issued when not in split mode without waiting for the associated write transfer to complete. However, because there are two holding registers, read transfers may only get one transfer ahead of write transfers.

4.11.4 Performance

The DMA can perform element transfers with single cycle throughput, if it accesses separate resources for the read transfer and write transfer, and both these resources have single-cycle throughput. An example of this is an un-synchronized block transfer from single-cycle external SBSRAM to internal data memory without any competition from any other channels or the CPU. The DMA performance can be limited by:

- The throughput and latency of the resources it requests.
- Waiting for read, write, or frame synchronization.
- Interruptions by higher priority channels.
- Contention with the CPU for resources.

4.12 DMA Action Complete Pins

The DMA action complete pins provide a method of feedback to external logic by generating an event for each channel (DMAC0–DMAC3). If specified by the DMAC EN bitfield in the DMA channel secondary control register, this pin can reflect the status of RSYNC STAT, WSYNC STAT, BLOCK COND, or FRAME COND, or be treated as a high or low general purpose output. If the DMAC bit reflects RSYNC STAT or WSYNC STAT, externally, then, once a synchronization event has been recognized, DMAC will transition from low to high. Once that same event has been serviced as indicated by the status bit being cleared, DMAC will transition from high to low. Before being sent off chip, the DMAC signals are synchronized by CLKOUT2 (1/2 the CPU clock rate). The active period of these signals is guaranteed to be a minimum of 2 CLKOUT2 periods wide.

4.13 Emulation

When using the emulator for debugging, the CPU may be halted on an execute packet boundary for single stepping, benchmarking, profiling, or other debugging purposes. The user may configure whether the DMA pauses during this time or continues running. This configuration is accompanied by setting the EMOD bit in the DMA primary control register to 0 or 1. If paused, the STATUS field will reflect the paused state of the channel. The auxiliary channel continues running during an emulation halt.

Chapter 5

Host-Port Interface

Describes the host-port interface for access of the ’C6201 memory-map space by external processors.

Topic **Page**

5.1 Overview

The host-port interface (HPI) is a 16-bit wide parallel port through which a host processor can directly access the CPU's memory space. The host device functions as a master to the interface, which increases its ease of access. The host and CPU can exchange information via internal or external memory. The host also has direct access to memory mapped peripherals. Connectivity to the CPU's memory space is provided through the DMA controller. Dedicated address and data registers not accessible to the CPU connect the HPI to the DMA auxiliary channel which connects the HPI to the CPU's memory space. The HPI and CPU can access the HPI control register (HPIC). The host can access the host address register (HPIA) and host data register (HPID) as well as the HPIC using the external data and interface control signals (Figure 5–1).

Figure 5–1. TMS320C6x Block Diagram

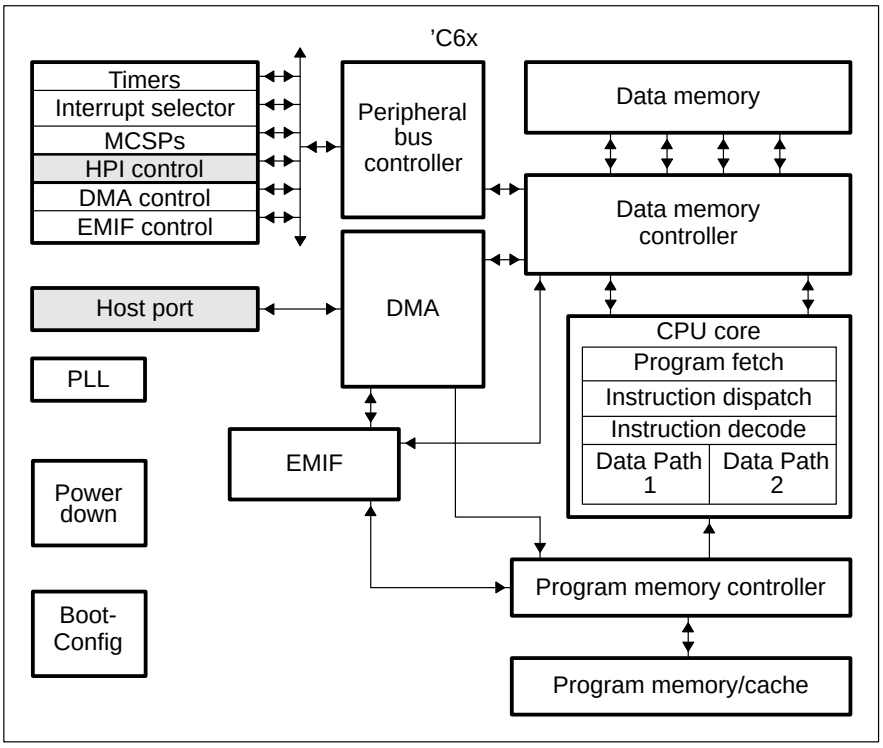


Figure 5–2. HPI Block Diagram

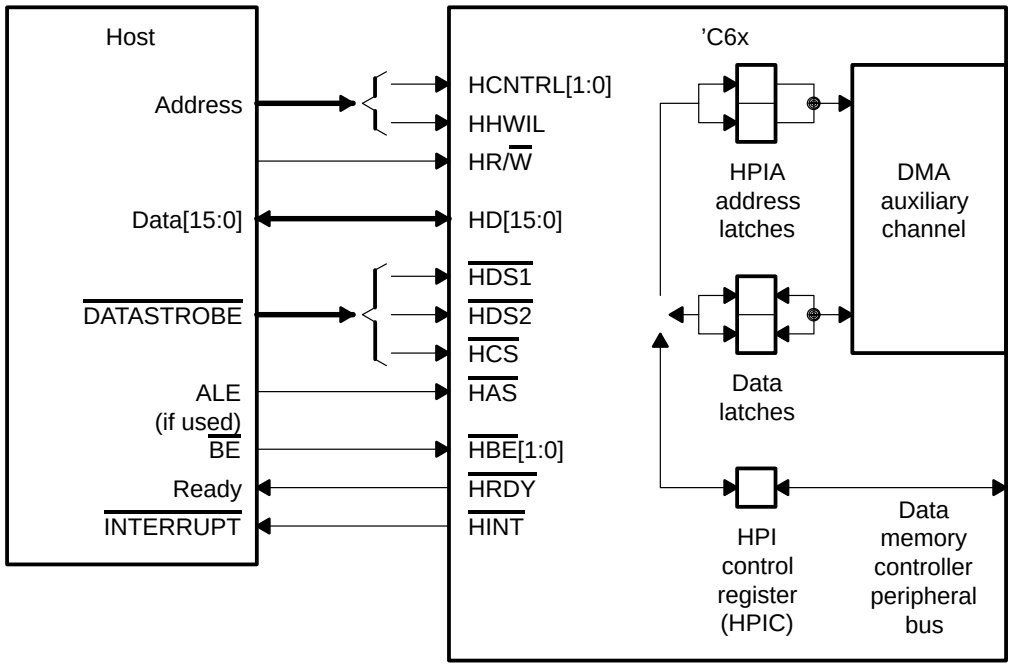


Figure 5–2 shows a simplified diagram of host interface to the HPI.

The HPI provides 32-bit data to the CPU with an economical 16-bit external interface by automatically combining successive 16-bit transfers. When the host device transfers data through HPID, the DMA auxiliary channel accesses the CPU's address space. The HPI supports high speed consecutive host accesses. In the absence of contention for memory mapped-resources, the HPI can transfer one 16-bit value every 4 CPU clocks. Thus, the DMA auxiliary channel can perform one 32-bit access every 8 CPU clocks.

The external HPI interface consists of the HPI data bus and control signals that configure and control the interface. The interface can connect to a variety of host devices with little or no additional logic.

The 16-bit data bus (HD0–HD15) exchanges information with the host. Because of the 32-bit word structure of the chip architecture, all transfers with a host consist of two consecutive 16-bit halfwords. On host data (HPID) write accesses, the $\overline{\text{HBE}}$[1:0] byte enables select which bytes in a 32-bit word should be written. HPIA, HPIC, and HPID read accesses are performed as 32-bit accesses, and the byte enables are not used. The dedicated HHWIL pin indicates whether the first or second halfword is being transferred. An internal control register bit determines whether the first or second halfword is placed into the most significant halfword

of a word. The host must not break the first halfword/second halfword (HHWIL low/high) sequence of an ongoing HPI access.

The two data strobes ($\overline{\text{HDS1}}$ and $\overline{\text{HDS2}}$), the read/write select (HR/$\overline{\text{W}}$), and the address strobe ($\overline{\text{HAS}}$) enable the HPI to interface to a variety of industry-standard host devices with little or no additional logic. The HPI can easily interface to hosts with a multiplexed or dedicated address/data bus, data strobe and a read/write strobe, or two separate strobes for read and write.

The HCNTL[1:0] control inputs indicate which HPI register is accessed. Using these inputs, the host can specify an access to the HPIA (which serves as the pointer into the source or destination space), HPIC, or HPID. These inputs, along with HHWIL, are commonly driven directly by host address bus bits or a function of these bits. the host can interrupt the CPU by writing to the HPIC; the CPU can activate $\overline{\text{HINT}}$ interrupt output to the host.

The host can access HPID with an optional automatic address increment of HPIA. This feature facilitates reading and writing to sequential word locations. In addition, auto-increment HPID reads prefetch the data at the auto-incremented access to reduce latency on the subsequent host read request.

The HPI ready pin ($\overline{\text{HRDY}}$) allows insertion of host wait-states. Wait-states may be necessary depending on latency to the point in the memory map accessed via the HPI as well as on the rate of host access. The rate of host access may force not-ready conditions if the host attempts to access the host port before any previous HPID write access or prefetched HPID read access completes. In this case, the HPI simply holds off the host via $\overline{\text{HRDY}}$. $\overline{\text{HRDY}}$ provides a convenient way to automatically (no software handshake needed) adjust the host access rate to the rate of data delivery from the DMA auxiliary channel. In the cases of hardware systems that cannot take advantage of the $\overline{\text{HRDY}}$ pin, a $\overline{\text{HRDY}}$ bit in the HPIC is available for use as a software handshake.

5.2 HPI Signal Description

The external HPI interface signals implement a flexible interface to a variety of host devices. Table 5–1 lists the HPI pins and their functions. The remainder of this section discusses the pins in detail.

Table 5–1. HPI External Interface Signals

Signal Name	Signal Type (Input/Output/ HI-Z)	Signal Count	Host Connection	Signal Function
HD[15:0]	I/O/Z	16	Data bus	
HCNTL[1:0]	I	2	Address or control lines	Controls HPI access type.
HHWIL	I	1	Address or control lines	Halfword identification input.
$\overline{\text{HAS}}$	I	1	Address latch enable (ALE) or address strobe or unused (tied high)	Differentiates address versus data values on multiplexed address/data host.
$\overline{\text{HBE}}$[1:0]	I	2	Byte enables	Data write byte enables
HR/$\overline{\text{W}}$	I	1	Read/write strobe, address line, or multiplexed address/ data	Read/Write select
$\overline{\text{HCS}}$	I	1	Address or control lines	Data strobe inputs.
$\overline{\text{HDS1}}$ $\overline{\text{HDS2}}$	I	2	Read strobe and write strobe or data strobe	Data strobe inputs.
$\overline{\text{HRDY}}$	O	1	Asynchronous ready	Ready status of current HPI access
$\overline{\text{HINT}}$	O	1	Host interrupt input.	Interrupt signal to host

5.2.1 Data Bus: HD[15:0]

HD[15:0] is a parallel, bidirectional, 3-state data bus. HD is placed in high-impedance when not performing an HPI read access.

5.2.2 Access Control Select: HCNTL[1:0]

HCNTL[1:0] indicate which internal HPI register is being accessed. The states of these two pins select access to the HPI address (HPIA), HPI data (HPID), or HPI control (HPIC) registers. Additionally, the HPID register can be accessed with an optional automatic address increment. Table 5–2 describes the HCNTL0/1 bit functions.

Table 5–2. HPI Input Control Signals Function Selection Descriptions

HCNTL1	HCNTL0	Description
0	0	Host can read or write the HPI control register, HPIC.
0	1	Host can read or write HPID. HPIA is postincremented by a word address (4 byte addresses).
1	0	Host can read or write the address register, HPIA.
1	1	Host can read or write HPID. HPIA is not affected.

5.2.3 Halfword Identification Select: HHWIL

Identifies the first or second halfword of transfer but not the most significant or least significant halfword. The HWOB bit specifies the halfword in the HPIC register, described later in this chapter. HHWIL is low for the first halfword and high for the second halfword.

5.2.4 Byte Enables: $\overline{\text{HBE}}$[1:0]

On HPID writes, the value of $\overline{\text{HBE}}$[1:0] indicates which portions of the 32-bit word must be written. The value of $\overline{\text{HBE}}$[1:0] is not important on HPIA or HPIC accesses or on HPID reads. On HPID writes, $\overline{\text{HBE0}}$ enables the least significant byte in the halfword and $\overline{\text{HBE1}}$ enables the MSByte in the halfword. Table 5–3 lists the valid combinations of byte enables. For byte writes, only one $\overline{\text{HBE}}$ in either of the halfword accesses may be enabled active-low. For halfword data writes, both the $\overline{\text{HBE}}$s must be held active-low in either (but not both) halfword accesses. For word accesses, both $\overline{\text{HBE}}$ must be held active-low in both halfword accesses. No other combinations are valid. The selection of byte-enables and the endianness of the CPU (selected via the LENDIAN pin), determine the logical address implied by the access.

Table 5–3. Byte Enables for HPI Data Write Access

Data Write Type	HBE-1:0		Effective Logical Address LSBs (binary)	
	Second Write HHWIL = 1	First Write HHWIL = 0	Little Endian	Big Endian
Byte	11	10	00	11
Byte	11	01	01	10
Byte	10	11	10	01
Byte	01	11	11	00
halfword	11	00	00	10
halfword	00	11	10	00
Word	00	00	00	00

5.2.5 Read/Write Select: HR/$\overline{W}$

HR/$\overline{W}$ is the host read/write select input. Hosts must drive HR/$\overline{W}$ high to read and low to write HPI. Hosts without either a read/write select output or a read or write strobe can use an address line for this function.

5.2.6 Ready: $\overline{HRDY}$

When active-low, $\overline{HRDY}$ indicates that the HPI is ready for a transfer to be performed. When inactive-high, $\overline{HRDY}$ indicates that the HPI is busy completing the internal portion of a current read access of a previous HPID read prefetch or write access. $\overline{HCS}$ enables $\overline{HRDY}$; $\overline{HRDY}$ is always high when $\overline{HCS}$ is high.

5.2.7 Strobes: $\overline{HCS}$, $\overline{HDS1}$, $\overline{HDS2}$

$\overline{HCS}$, $\overline{HDS1}$, and $\overline{HDS2}$ allow connection to hosts with either:

- Single strobe output with read/write select.
- Separate read and write strobe outputs. In this case, read or write select can be done by using different addresses.

Figure 5–3 shows the equivalent circuit of the $\overline{\text{HCS}}$, $\overline{\text{HDS1}}$, and $\overline{\text{HDS2}}$ inputs.

Used together, $\overline{\text{HCS}}$, $\overline{\text{HDS1}}$, and $\overline{\text{HDS2}}$ generate an active-low internal $\overline{\text{HSTRB}}$ signal. Note that $\overline{\text{HSTRB}}$ is only active low when both $\overline{\text{HCS}}$ is active-low and either (but not both) $\overline{\text{HDS1}}$ and $\overline{\text{HDS2}}$ are active low. The falling edge of $\overline{\text{HSTRB}}$ (when $\overline{\text{HAS}}$ is tied inactive-high) samples HCNTL[1:0], HHWIL, HR/$\overline{\text{W}}$, and $\overline{\text{HBE}}$[1:0]. Therefore, whichever of $\overline{\text{HDS1}}$, $\overline{\text{HDS2}}$, or $\overline{\text{HCS}}$ that falls the latest controls the sampling time. $\overline{\text{HCS}}$ serves as the enable input for the HPI and must be low during an access. However, as the $\overline{\text{HSTRB}}$ signal determines the actual boundaries between accesses, $\overline{\text{HCS}}$ may stay low between successive accesses as long as both $\overline{\text{HDS1}}$ and $\overline{\text{HDS2}}$ go inactive-high or go active-low.

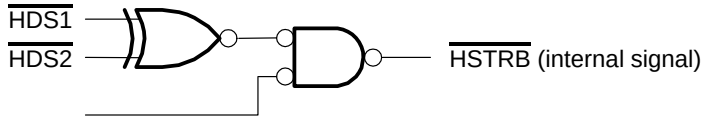
Hosts with separate read and write strobes connect these strobes to either $\overline{\text{HDS1}}$ or $\overline{\text{HDS2}}$. Hosts with a single data strobe connect it to either $\overline{\text{HDS1}}$ or $\overline{\text{HDS2}}$, connecting to the unused pin high. Regardless of HDS connections, the HR/$\overline{\text{W}}$ is still required to determine direction of transfer. Because $\overline{\text{HDS1}}$ and $\overline{\text{HDS2}}$ are internally exclusive-NORed, hosts with a high true data strobe can connect this to one of the HDS inputs with the other $\overline{\text{HDS}}$ input connected low.

$\overline{\text{HSTRB}}$ is used for three purposes:

1) On a read, the falling edge of $\overline{\text{HSTRB}}$ initiates HPI read accesses for all access types.
2) On a write, its rising edge initiates HPI write accesses for all access types.
3) Falling edges latch HPI control inputs including: HHWIL, HR/$\overline{\text{W}}$, $\overline{\text{HBE}}$[1:0], HCNTL[1:0]. $\overline{\text{HAS}}$ also effects latching of control inputs. See subsection 5.2.8 for a description of $\overline{\text{HAS}}$.

Additionally, $\overline{\text{HCS}}$ gates the $\overline{\text{HRDY}}$ output. In other words, a not-ready condition is indicated by the $\overline{\text{HRDY}}$ pin driven low only if $\overline{\text{HCS}}$ is active-low. Otherwise $\overline{\text{HRDY}}$ is inactive(high).

Figure 5–3. Select Input Logic



5.2.8 Address Strobe Input: $\overline{\text{HAS}}$

$\overline{\text{HAS}}$ allows $\overline{\text{HCNTL}}$[1:0], $\overline{\text{HBE}}$[1:0], HR/$\overline{\text{W}}$, and HHWIL to be removed earlier in an access cycle, which allows more time to switch bus states from address to data information. This feature facilitates interface to multiplexed address and data type buses. In this type of system, an ALE signal is often provided and would normally be the signal connected to HAS.

Hosts with a multiplexed address and data bus connect HAS to their ALE pin or an equivalent pin. HHWIL, HCNTL0/1, $\overline{\text{HBE}}$,and HR/$\overline{\text{W}}$ are latched on $\overline{\text{HAS}}$'s falling edge. When used, $\overline{\text{HAS}}$ must precede the latest of $\overline{\text{HCS}}$, $\overline{\text{HDS1}}$, or $\overline{\text{HDS2}}$. Hosts with separate address and data bus can tie $\overline{\text{HAS}}$ high. In this case, HHWIL, HCNTL0/1, and HR/$\overline{\text{W}}$ are latched by the later of $\overline{\text{HDS1}}$, $\overline{\text{HDS2}}$, or $\overline{\text{HCS}}$ falling edge while $\overline{\text{HAS}}$ stays inactive (high).

5.2.9 Interrupt to Host: $\overline{\text{HINT}}$

$\overline{\text{HINT}}$ is the host interrupt output which is controlled by the $\overline{\text{HINT}}$ bit in the HPIC. This bit is driven inactive-high when the chip is being reset. This signal is described in more detail in subsection 5.3.5.

5.2.10 HPI Bus Access

Figure 5–5 and Figure 5–4 show HPI access timing for the cases when $\overline{\text{HAS}}$ is and is not used, respectively. $\overline{\text{HSTRB}}$ represents the internally generated strobe described in Figure 5–3. HAD represents control signals typically driven by host address lines: HCNTL1:0, HR/$\overline{\text{W}}$, HHWIL, and $\overline{\text{HBE}}$[1:0]. HCNTL[1:0] and HR/$\overline{\text{W}}$ should have the same values for both halfword accesses. HHWIL is shown separately to indicate that it must be low for the first halfword transfer and high for the second. If HAS is not used (tied high as shown in Figure 5–4), the falling edge of $\overline{\text{HSTRB}}$ latches these signals. If HAS is used as shown in Figure 5–5, the falling edge of $\overline{\text{HAS}}$ latches these values. In this case, the falling edge of $\overline{\text{HAS}}$ must precede the falling edge of $\overline{\text{HSTRB}}$. On a read, data is valid a delay after the falling edge of $\overline{\text{HSTRB}}$. If valid data is not already present in the HPID, the data is setup at the rising edge of $\overline{\text{HRDY}}$ and held until the rising edge of $\overline{\text{HSTRB}}$. On a write, the host must setup data to the rising edge of $\overline{\text{HSTRB}}$.

Figure 5–4. HPI Timing Diagram Using HAS

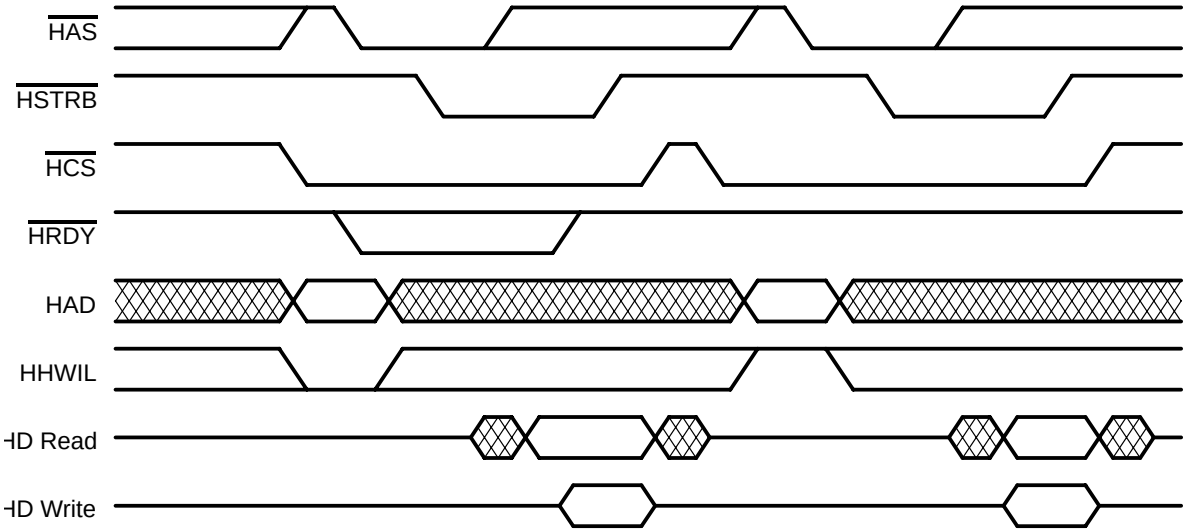
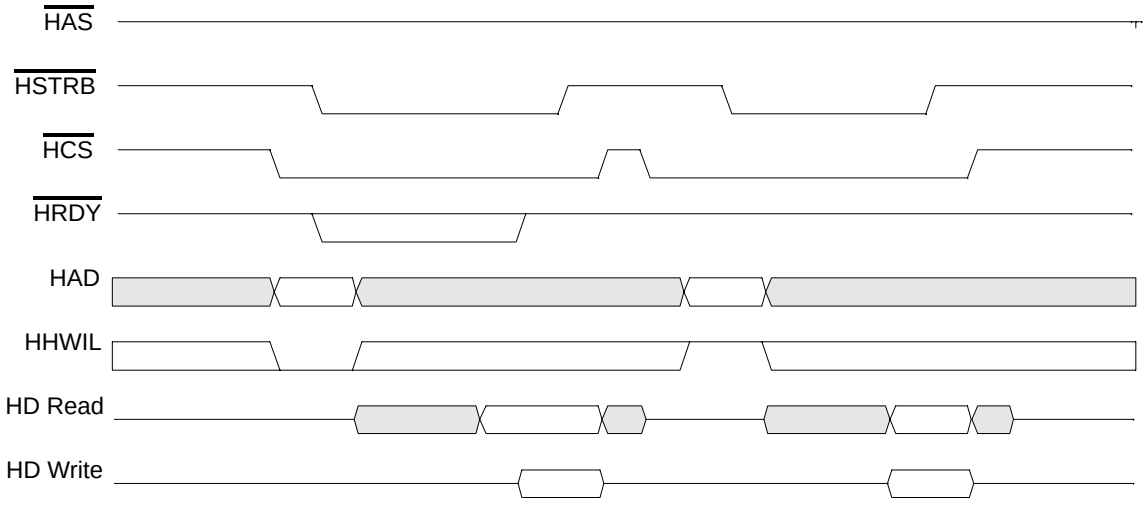


Figure 5–5. HPI Timing Diagram Not Using HAS



5.3 HPI Registers

Table 5–4 summarizes the three registers that the HPI utilizes for communication between the host device and the CPU. HPID contains the data that was read from the HPI memory if the current access is a read, or the data that will be written to HPI memory if the current access is a write. HPIA contains the address in the HPI memory at which the current access occurs. As this address is a 30-bit word address, the bottom two bits are unaffected by HPIA writes and are always read as 0.

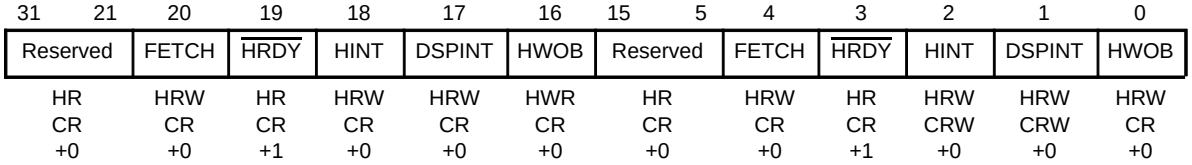
Table 5–4. HPI Register Description

Register Acronym	Register Name	Host Read/Write	CPU Read/Write	CPU Read/Write (Hex Byte Address)
HPID	HPI data	RW	–	–
HPIA	HPI address	RW	–	–
HPIC	HPI control	RW	RW	0188 0000h

5.3.1 HPI Control Register (HPIC)

The HPIC diagrammed in Figure 5–6 is normally the first register accessed when setting configuration bits and initialize the interface. Thus, the HPIC is organized on the as a 32-bit register with the same high and low halfword contents. On a host write, both halfwords must be identical. Note that the low halfword and the high halfword are actually the same storage locations. No storage is allocated for the read only reserved values. Only CPU writes to the lower halfword affect HPIC values and HPI operation. The HPIC is normally the first register accessed to set configuration bits and initialize the interface. Thus, the HPIC is organized as a 32-bit register with the same high and low halfword contents.

Figure 5–6. HPIC Register Diagram



Note: CR = CPU Read, CRW = CPU Read/Write, HR = Host Read, HRW = Host Read/Write

Table 5–5. HPI Control Register (HPIC) Bit Descriptions

Bit	Description	Subsection
HWOB	Halfword ordering bit If HWOB = 1, first halfword is least significant. If HWOB = 0, first halfword is most significant. HWOB affects both data and address transfers. Only the host can modify this bit. HWOB must be initialized before the first data or address register access.	5.4
DSPINT	The host processor-to-CPU/DMA interrupt.	5.3.4
HINT	DSP to Host Interrupt. Determines the state of the CPU $\overline{\text{HINT}}$ output.	5.3.5
HRDY	Ready signal to host. Not masked by $\overline{\text{HCS}}$ like $\overline{\text{HRDY}}$ pin. If $\overline{\text{HRDY}}$ = 0 this indicates that the internal bus is waiting for a HPI data access request to complete.	5.3.2
FETCH	Host Fetch request. If the host writes a one to this bit, it requests a fetch into HPID the word at the address pointed to by HPIA. Always read as 0.	5.3.2

5.3.2 Software Handshaking Using HRDY and FETCH

As described previously, the $\overline{\text{HRDY}}$ pin can indicate to a host that a HPID access has not completed. For example, the current HPID access can be waiting for a previous HPID access write to complete or for a previous HPID prefetched read to complete. Also, the current HPID read access can be waiting for its requested data to arrive. However, the $\overline{\text{HRDY}}$ and FETCH bits in the HPIC allow for a software handshake that allows an HPI connection in systems where a hardware ready control is not desired. The FETCH and $\overline{\text{HRDY}}$ bits can be used to perform a read transfer as follows:

1) The host polls for the $\overline{\text{HRDY}}$ bit to be set.
2) The host writes the desired HPIA value. This step is skipped if HPIA is already set to the desired value.
3) The host writes a 1 to the FETCH bit.
4) The host polls for the $\overline{\text{HRDY}}$ bit to be set.
5) The host performs a HPID read operation. In this case, the HPI is already in the ready state ($\overline{\text{HRDY}}$ bit = 1).
6) If this was a read with post increment, go to 4.

   For a read from the same location, go to 3.

   For a read to a different address, go to 2.

The $\overline{\text{HRDY}}$ bit alone can be used for write operations as follows:

1) The host polls for the $\overline{\text{HRDY}}$ bit to be set.
2) The host writes the desired HPIA value. (This step is skipped if HPIA is already set to the desired value.)
3) The host performs a HPID write operation.

   For another write operation, go to 1.

5.3.3 DSPINT and HINT Function Operation

The host and the CPU can interrupt each other using bits in the HPIC register. This subsection presents more information about this process.

5.3.4 Host Device Using DSPINT to Interrupt the CPU

The host can interrupt the CPU by writing to the DSPINT bit in the HPIC. The DSPINT bit is tied directly to the internal DSPINT signal. By writing DSPINT = 1 when DSPINT = 0, the host causes a 0-to-1 transition on the DSPINT signal. If appropriately selected by the interrupt selector, this transition will be detected as an interrupt condition by the CPU. The CPU can clear the DSPINT bit by writing a 1 to DSPINT. Neither a host nor a CPU HPIC write with DSPINT = 0 affects the DSPINT bit or signal.

5.3.5 CPU Using $\overline{\text{HINT}}$ to Interrupt the Host

The CPU can send an active low interrupt condition on the $\overline{\text{HINT}}$ signal by writing to the HINT bit in the HPIC. The HINT bit is inverted and tied directly to the $\overline{\text{HINT}}$ pin. The CPU can set $\overline{\text{HINT}}$ active low by writing HINT = 1. The host can clear the $\overline{\text{HINT}}$ to inactive-high bit by writing DSPINT = 1. Neither a host nor a CPU HPIC write with HINT = 0 effects either the HINT bit or $\overline{\text{HINT}}$ signal. Note that this bit is read twice on the host interface side, the first and second halfword reads by the host may yield different data if the CPU changes the state of one or both of these bits between the two read operations.

5.4 Host Access Sequences

The host begins HPI accesses by initializing the HPIC, then HPIA, and then writing data to or reading data from HPID. Reading or writing HPID initiates an internal cycle that transfers the desired data between the HPID and the DMA auxiliary channel. Host access of any HPI register requires two halfword accesses on the HPI bus: the first with HHWIL low, and the second with HHWIL high. Typically, the host does not break the first halfword/second halfword (HHWIL low/high) sequence. If this sequence is broken improperly, data may be lost, and undesired operation may result. The first halfword access may have to wait for a previous HPI request to complete. Previous requests include HPID writes and pre-fetched HPID reads. Thus, the HPI will deassert $\overline{\text{HRDY}}$ high until the HPI can begin this request. The second halfword access will always have $\overline{\text{HRDY}}$ active low since all previous accesses have been completed for the first halfword access.

5.4.1 Host Initialization of HPIC and HPIA

Before accessing data, the host must first initialize the HWOB bit of the HPIC and then HPIA (in this order, because HWOB affects the HPIA access). After initializing HWOB, the host can then write to HPIA with the correct halfword alignment. Table 5–6 and Table 5–7 illustrate the initialization sequence for HWOB = 1 and HWOB = 0, respectively. In these examples, HPIA is set to 80001234h. Note that in all these accesses, $\overline{\text{HRDY}}$ bit in the HPIC is set (bit 19 and 3).

Table 5–6. Initialization of HWOB = 1 and HPIA

Event	Value During Access					Value After Access		
	HD	$\overline{\text{HBE[1:0]}}$	HR/$\overline{\text{W}}$	HCNTL[1:0]	HHWIL	HPIC	HPIA	HPID
Host writes HPIC 1st halfword	0001	xx	0	00	0	00090009	????????	????????
Host writes HPIC 2nd halfword	0001	xx	0	00	1	00090009	????????	????????
Host writes HPIA 1st halfword	1234	xx	0	01	0	00090009	????1234	????????
Host writes HPIA 2nd halfword	8000	xx	0	01	1	00090009	80001234	????????

Table 5–7. Initialization of HWOB = 0 and HPIA

Event	Value During Access					Value After Access		
	HD	$\overline{\text{HBE}}$[1:0]	HR/$\overline{\text{W}}$	HCNTL[1:0]	HHWIL	HPIC	HPIA	HPID
Host writes HPIC 1st halfword	0000	xx	0	00	0	00080008	????????	????????
Host writes HPIC 2nd halfword	0000	xx	0	00	1	00080008	????????	????????
Host writes HPIA 1st halfword	8000	xx	0	01	0	00080008	8000????	????????
Host writes HPIA 2nd halfword	1234	xx	0	01	1	00080008	80001234	????????

5.4.2 HPID Read Access Without Auto-Increment

Assume that once the HPI is initialized that the host wishes to do a read access to that address without an autoincrement. Assume that the host wants to read the word at address 80001234h and that the word value at that location is 789ABCDEh. Table 5–8 and Table 5–9 illustrate this access for HWOB = 1 and HWOB = 0, respectively. On the first halfword accesses, the HPI waits for any previous requests to complete. During this time, $\overline{\text{HRDY}}$ is held inactive-high. Then, the HPI sends the read request to the DMA auxiliary channel. If no previous requests are pending, this event occurs with the falling edge of $\overline{\text{HSTRB}}$. $\overline{\text{HRDY}}$ remains low until the DMA auxiliary channel loads the requested data into HPID. Because all DMA auxiliary channel reads are word reads, at the beginning of the second read access the data is already present in HPID. Thus, the second halfword HPID read will never encounter a not ready condition and $\overline{\text{HRDY}}$ will remain active-low. The byte enables are not important as the HPI only does word reads.

Table 5–8. Read Access to HPI Without Autoincrement: HWOB = 1

Event	Value During Access						Value After Access		
	HD	$\overline{\text{HBE}}$[1:0]	HR/$\overline{\text{W}}$	HCNTL[1:0]	$\overline{\text{HRDY}}$	HHWIL	HPIC	HPIA	HPID
Host reads 1st halfword Data not Ready	????	xx	1	11	1	0	00010001	80001234	????????
Host reads 2nd half-word Data ready	BCDE	xx	1	11	0	0	00090009	80001234	789ABCDE
Host reads 2nd half-word	789A	xx	1	11	0	1	00090009	80001234	789ABCDE

Table 5–9. Read Access to HPI Without Autoincrement: HWOB = 0

	Value During Access						Value After Access		
Event	HD	$\overline{\text{HBE}}$[1:0]	HR/$\overline{\text{W}}$	HCNTL[1:0]	$\overline{\text{HRDY}}$	HHWIL	HPIC	HPIA	HPID
Host reads 1st halfword Data not Ready	????	xx	1	11	1	0	00000000	80001234	????????
Host reads 2nd half-word Data Ready	798A	xx	1	11	0	0	00080008	80001234	789ABCDE
Host Reads 2nd half-word	BCDE	xx	1	11	0	1	00080008	80001234	789ABCDE

5.4.3 HPID Read Access With Auto-Increment

The autoincrement feature results in efficient sequential host accesses. For both HPID read and write accesses, this removes the need for the host to load incremented address into HPIA. For read accesses, the data pointed to by the next address is fetched immediately after the completion of the current read. Because the intervals between successive reads is used to prefetch data, the latency for the next access is reduced. Prefetching can also occur after a host write of FETCH = 1 to the HPIC. If the next HPI access is a HPID read, then the data is not re-fetched and the pre-fetched data is sent to the host. Otherwise, the HPI must still wait for the prefetch to complete.

Table 5–10 shows a read access with auto-increment.

After the first halfword access is complete (with the rising edge of the first $\overline{\text{HSTRB}}$), the address increments to the next word or 80001238h. Assume that the data at that location is 87654321h. This data is pre-fetched and loaded into HPID. Prefetching begins on the rising edge of $\overline{\text{HSTRB}}$ on the second halfword read.

Table 5–10. Read Access to HPI With Autoincrement: HWOB = 1

Event	Value During Access						Value After Access		
	HD	$\overline{\text{HBE}}$[1:0]	HR/$\overline{\text{W}}$	HCNTL[1:0]	$\overline{\text{HRDY}}$	HHWIL	HPIC	HPIA	HPID
Host Reads 1st halfword Data Not Ready	????	xx	1	10	1	0	00010001	80001234	????????
Host Reads 2nd halfword Data Ready	BCDE	xx	1	10	0	0	00090009	80001234	789ABCDE
Host Reads 2nd halfword	789A	xx	1	10	0	1	00090009	80001238	789ABCDE
Pre-fetch Data Not Read	????	xx	x	xx	0	x	00010001	80001238	789ABCDE
Pre-fetch Data Ready	????	xx	x	xx	0	x	00090009	80001238	87654321

Table 5–11. Read Access to HPI With Autoincrement: HWOB = 0

Event	Value During Access						Value After Access		
	HD	$\overline{\text{HBE}}$[1:0]	HR/$\overline{\text{W}}$	HCNTL[1:0]	$\overline{\text{HRDY}}$	HHWIL	HPIC	HPIA	HPID
Host Reads 1st halfword Data Not Ready	????	xx	1	10	1	0	00000000	80001234	????????
Host Reads 2nd halfword Data Ready	789A	xx	1	10	0	0	00080008	80001234	789ABCDE
Host Reads 2nd halfword	BCDE	xx	1	10	0	1	00080008	80001238	789ABCDE
Pre-fetch Data Not Read	????	xx	x	xx	0	x	00000000	80001238	789ABCDE
Pre-fetch Data Ready	????	xx	x	xx	0	x	00080008	80001238	87654321

5.4.4 Host Data Write Access Without Auto Increment

During a write access to the HPI, the first halfword portion of HPID (LShalfword or MShalfword as selected by HWOB) is overwritten by the data coming from the host and the first $\overline{\text{HBE}}$[1:0] latched while the HHWIL pin is low. The second halfword portion of HPID is overwritten by the data coming from the host and the second $\overline{\text{HBE}}$[1:0] pair is latched while the HHWIL pin is high. At the end of this write access (with the second rising edge $\overline{\text{HSTRB}}$), HPID is transferred as a 32-bit word to the address specified by HPIA with the four related byte-enables.

Table 5–12 and Table 5–13 illustrate an HPID write access with HWOB = 1 and HWOB = 0, respectively. The host writes 5566h to the 16 LSBs of location 80001234h, which is already pointed to by HPIA. This location is assumed to start with the value 0. The HPI holds off the host until any previous transfers are completed by setting $\overline{\text{HRDY}}$ inactive-high. Also, once the first halfword becomes ready, the second halfword does not encounter any not ready time. If there are no pending writes waiting in HPID, then write accesses normally proceed without a not ready time. Note that the $\overline{\text{HBE}}$[1:0] is only enabled for the transfer which transfers the 16 LSBs.

Table 5–12. Write Access to HPI Without Autoincrement: HWOB = 1†

Event	Value During Access						Value After Access			Location 80001234
	HD	$\overline{\text{HBE}}$[1:0]	HR/$\overline{\text{W}}$	HCNTL[1:0]	$\overline{\text{HRDY}}$	HHWIL	HPIC	HPIA	HPID	
Host writes HPID 1st halfword Waiting for Previous	5566	00	0	11	1	0	00010001	80001234	????5566	00000000
Host writes HPID 1st halfword Ready	5566	00	0	11	0	0	00090009	80001234	????5566	00000000
Host writes HPIA 2nd halfword	wxyz	11	0	11	0	1	00090009	80001234	wxyz5566	00005566

† wxyz represents a don't care value on the HD pins.

Table 5–13. Write Access to HPI Without Autoincrement: HWOB = 0

Event	Value During Access						Value After Access			Location 80001234
	HD	$\overline{\text{HBE}}$[1:0]	HR/$\overline{\text{W}}$	HCNTL[1:0]	$\overline{\text{HRDY}}$	HHWIL	HPIC	HPIA	HPID	
Host writes HPID 1st halfword Waiting for Previous	wxyz	11	0	11	1	0	00000000	80001234	wxyz????	00000000
Host writes HPID 1st halfword Ready	wxyz	11	0	11	0	0	00080008	80001234	wxyz????	00000000
Host writes HPIA 2nd halfword	5566	00	0	11	0	1	1 00080008	80001234	wxyz5566	00005566

5.4.5 HPID Write Access with Auto-Increment

Table 5–14 and Table 5–15 show a host write data with auto-increment for HWOB = 1 and HWOB = 0, respectively. These examples are identical to the ones in subsection 5.4.4, with the exception of the HCNTL[1:0] value and a subsequent write of 33 to the most significant byte of the word at address 80001238. The increment happens on the rising edge of $\overline{\text{HSTRB}}$ on the next HPID write access. If the next access is an HPID or HPIC access, or a HPID read, the autoincrement does not occur.

Table 5–14. Write Access to HPI With Autoincrement: HWOB = 1††

Event	Value During Access						Value After Access			Location 80001234	Location 80001238
	HD	$\overline{\text{HBE}}$ [1:0]	HR/$\overline{\text{W}}$	HCNTL [1:0]	$\overline{\text{HRDY}}$	HHWIL	HPIC	HPIA	HPID		
Host writes HPID 1st halfword Waiting for Previous	wxyz	11	0	11	1	0	00000000	80001234	wxyz????	00000000	00000000
Host writes HPID 1st halfword Waiting for Previous	5566	00	0	01	1	0	00010001	80001234	????5566	00000000	00000000
Host writes HPID 1st halfword Ready	5566	00	0	01	0	0	00090009	80001234	????5566	00000000	00000000
Host writes HPIA 2nd halfword	wxyz	11	0	01	0	1	00090009	80001234	wxyz5566	00005566	00000000
Host writes HPID 1st halfword Waiting for Previous	nopq	11	0	01	1	0	00010001	80001234	wxyznopq	00005566	00000000
Host writes HPID 1st halfword Ready	nopq	11	0	01	0	1	00090009	80001238	wxyznopq	00005566	00000000
Host writes HPIA 2nd halfword	33rs	01	0	01	1	1	00090009	80001238	33rsnopq	00005566	33000000

†† wxyz, rs, and nopd represent don’t care values on HPID.

Table 5–15. Write Access to HPI Without Autoincrement: HWOB = 0†

	Value During Access						Value After Access				
Event	**HD**	**$\overline{\text{HBE}}$ [1:0]**	**HR/$\overline{\text{W}}$**	**HCNTL [1:0]**	**$\overline{\text{HRDY}}$**	**HHWIL**	**HPIC**	**HPIA**	**HPID**	**Location 80001234**	**Location 80001238**
Host writes HPID 1st halfword Waiting for Previous	5566	00	0	01	1	0	00000000	80001234	????5566	00000000	00000000
Host writes HPID 1st halfword Ready	5566	00	0	01	0	0	00080008	80001234	????5566	00000000	00000000
Host writes HPID 2nd halfword	wxyz	11	0	01	0	1	00080008	80001234	wxyz5566	00005566	00000000
Host writes HPID 1st halfword Waiting for Previous	33rs	11	0	01	1	0	00000000	80001234	33rs5566	00005566	00000000
Host writes HPID 1st halfword Ready	33rs	11	0	01	0	1	00080008	80001238	33rs5566	00005566	00000000
Host writes HPID 2nd halfword	nopq	01	0	01	0	1	00080008	80001238	33rsnopq	00005566	33000000

† wxyz, rs, and nopd represent don't care values on HPID.

5.4.6 Single Half-word Cycles

In normal operation, every transfer must consist of two halfword accesses. However, to speed operation the user may perform single halfword accesses. These can be useful in several cases:

- HPIC: Note that in Table 5–6 that the entire HPIC was written correctly after the first write. When writing the HPIC, the host does not have to be concerned about HHWIL, nor does it have to perform two consecutive writes to both halfwords. Similarly, the host can choose to only read the HPIC once since both halves contain the same value.

- HPIA: Note that in Table 5–6, the portion of HPIA accesses as selected by HHWIL and HWOB automatically is updated after each halfword access. Thus, to change the only the upper or lower 16 bits of HPIA the host only needs to select which half to modify through a combination of HHWIL and HWOB. Similarly, the host can choose to only read the desired half of HPIA.

- HPID read accesses: Read accesses are actually triggered by the first halfword access (HHWIL low). Thus, if on reads the host is only interested in the first halfword (least or most significant selected by HWOB), the host does not need to request the second address. However, pre-fetching will not occur unless the second halfword is also read. A subsequent read of the first halfword (HHWIL low) or the write of a new value to HPIA will override any previous prefetch request. On the other hand, a read of just the second halfword (HHWIL high) is not allowed and can result in undefined operation.

- Write accesses: Write accesses are triggered by the second halfword access (HHWIL is high). Thus, if the host only desires to change the portion of HPID selected by HHWIL high (and the associated byte enables) during consecutive write accesses, only a single cycle would need to be initiated. This technique primary use would be for memory fills. To do this the host would write both halfwords of the first write access with HBE[1:0] = 00. On subsequent write accesses, the host would make sure it writes the same value to the portion of HPID selected by HHWIL as did the first write access. In this case, the host would perform auto-incrementing writes (HCNTL[1:0] = 01) on all write accesses.

5.5 Access of HPI Memory During Reset

The HPI cannot be used while the chip is in reset. However, certain boot modes may allow the host to write the CPU's memory space (including configuring EMIF configuration registers to define external memory before accessing it). Although the device is not in reset, the CPU itself is in reset until the boot completes. See Chapter 7, *Boot Configuration, Reset, and Memory Map*, for more details.

Chapter 6

External Memory Interface

Decribes the external memory interface used by the CPU to access off-chip memory.

Topic **Page**

6.1 Overview

The external memory interface (EMIF) supports a glueless interface to a variety external devices, including:

- Synchronous burst SRAM (SBSRAM) running at 1× and 1/2× the CPU clock rate
- Synchronous DRAM (SDRAM) running at 1/2× the CPU clock rate
- Asynchronous devices, including asynchronous SRAM, ROM, and FIFOs. The EMIF provides highly programmable timing to these interfaces.

The EMIF services requests of the external bus from four requesters as shown in Figure 6–1:

- The on-chip program memory controller that services CPU program fetches
- The on-chip data memory controller that services CPU data fetches
- The on-chip DMA controller
- An external shared-memory device

If multiple requests arrive simultaneously, the EMIF prioritizes them and performs the necessary cycles. A block diagram of the EMIF is shown in Figure 6–2. Table 6–1 describes the EMIF signals.

Figure 6–1. External Memory Interface in the TMS320C6x Block Diagram

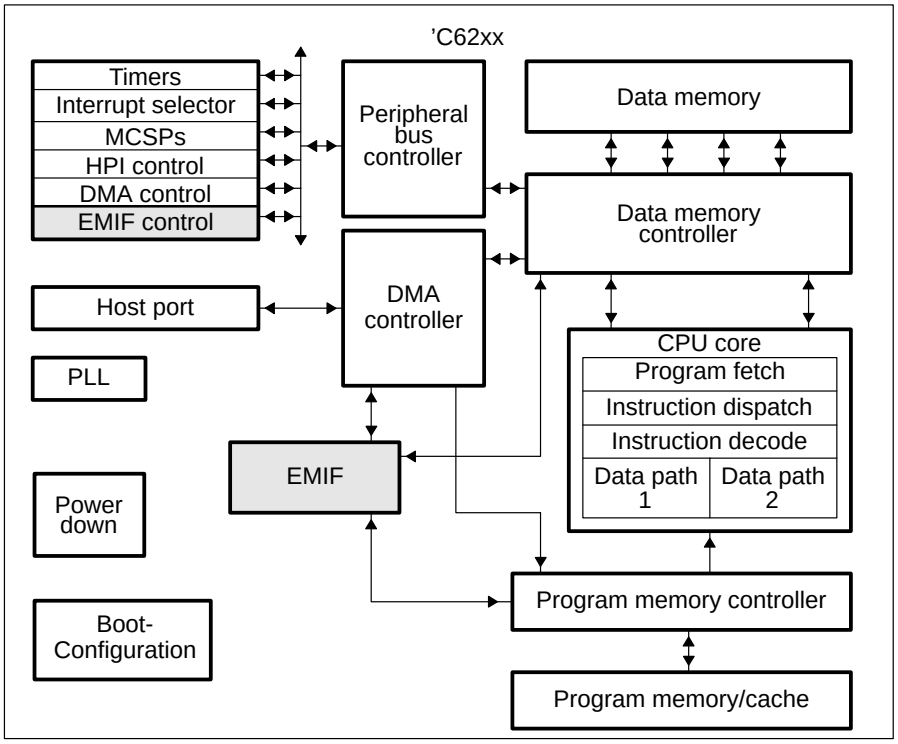


Figure 6–2. EMIF Block Diagram

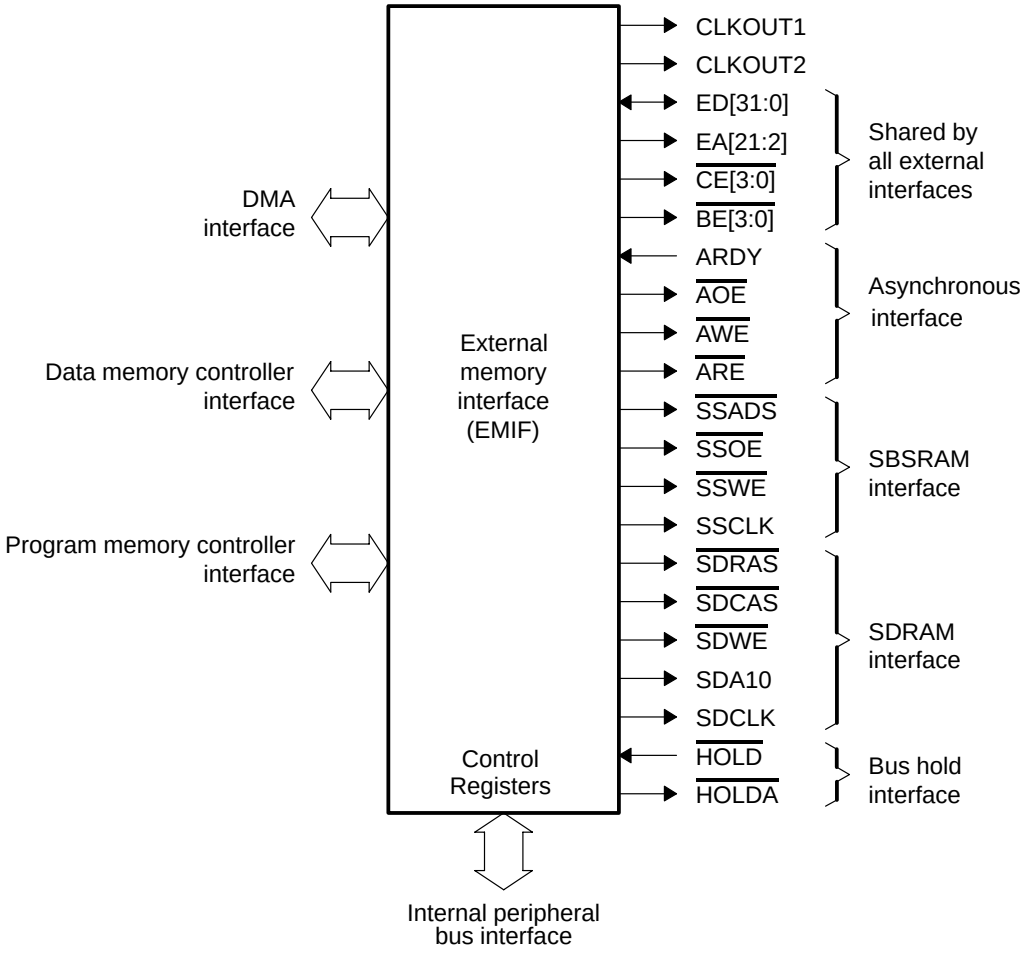


Table 6–1. EMIF Signal Descriptions

Pin	(I/O/Z)	Description
CLKOUT1	O	Clock output. The CPU clock rate
CLKOUT2	O	Clock output. 1/2 the CPU clock rate
ED[31:0]	I/O/Z	Data I/O. 32-bit data input/output from external memories and peripherals
EA[21:2]	O/Z	External address output. Drives bits 21-2 of the byte address.
$\overline{\text{CE}}$0	O/Z	Active-low chip select for memory space CE0
$\overline{\text{CE}}$1	O/Z	Active low chip select for memory space CE1
$\overline{\text{CE}}$2	O/Z	Active low chip select for memory space CE2
$\overline{\text{CE}}$3	O/Z	Active low chip select for memory space CE3
$\overline{\text{BE}}$(3:0)	O/Z	Byte enables. Active low byte enable. Individual bytes and halfwords can be selected for both read and write cycles. Decoded from two LSBs of the byte address.
ARDY	I	Ready. Active-high asynchronous ready input used to insert wait states for slow memories and peripherals
$\overline{\text{AOE}}$	O/Z	Active-low output enable for asynchronous memory interface
$\overline{\text{AWE}}$	O/Z	Active-low write strobe for asynchronous memory interface
$\overline{\text{ARE}}$	O/Z	Active-low read strobe for asynchronous memory interface
$\overline{\text{SSADS}}$	O/Z	Active-low address strobe/enable for SBSRAM interface
$\overline{\text{SSOE}}$	O/Z	Output buffer enable for SBSRAM interface
$\overline{\text{SSWE}}$	O/Z	Active-low write enable for SBSRAM interface
SSCLK	O/Z	SBSRAM interface clock. Equivalent to CLKOUT1 or CLKOUT2 as selected by the user
$\overline{\text{SDRAS}}$	O/Z	Active-low row strobe for SDRAM memory interface
$\overline{\text{SDCAS}}$	O/Z	Active-low column strobe for SDRAM memory interface
$\overline{\text{SDWE}}$	O/Z	Write enable. Active-low write enable for SDRAM memory interface
SDA10	O/Z	SDRAM A10 address line. Address line/auto precharge disable for SDRAM memory. Used to refresh SDRAM even while no SDRAM accesses are using the bus.
SDCLK	O/Z	SDRAM interface clock. 1/2 the CPU clock rate. Equivalent to CLKOUT2.
$\overline{\text{HOLD}}$	I	Active-low external bus hold (3-state) request
$\overline{\text{HOLDA}}$	O	Active-low external bus hold acknowledge

6.1.1 Resetting the EMIF

A hardware reset using the $\overline{\text{RESET}}$ pin on the device forces all register values to their reset state. During reset, all outputs are driven to their inactive levels, with the exception of the clock outputs (SDCLK, SSCLK, CLKOUT1, and CLKOUT2). CLKOUT2, SSCLK, and SDCLK are driven high or low during active $\overline{\text{RESET}}$. CLKOUT1 continues clocking unless the PLL configuration pins are changed or if the SSCLK configuration is different from reset SSCLK configuration.

6.2 EMIF Registers

Control of the EMIF and the memory interfaces it supports is maintained through memory-mapped registers within the EMIF. A write to any EMIF register is not complete until all pending EMIF accesses that use that register are completed.

The memory-mapped registers are shown in Table 6–2.

Table 6–2. EMIF Memory-Mapped Registers

Byte Address	Name
0x01800000	EMIF global control
0x01800004	EMIF CE1 space control
0x01800008	EMIF CE0 space control
0x0180000C	Reserved
0x01800010	EMIF CE2 space control
0x01800014	EMIF CE3 space control
0x01800018	EMIF SDRAM control
0x0180001C	EMIF SDRAM refresh period

6.2.1 EMIF Global Control Register

The EMIF global control register (shown in Figure 6–3 and described in Table 6–3) configures parameters common to all the CE spaces.

Figure 6–3. EMIF Global Control Register Diagram

31 14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	SDCINV	CLK2INV	Rsv	ARDY	$\overline{\text{HOLD}}$	$\overline{\text{HOLDA}}$	NOHOLD	SDCEN	SSCEN	CLK1EN	CLK2EN	SSCRT	RBTR8	MAP
R, +0	RW, +1	RW, +1	R, +0	R, +x	R, +x	R, +0	RW, +0	RW, +1	RW, +1	RW, +1	RW, +1	RW, +0	RW, +0	R, +x

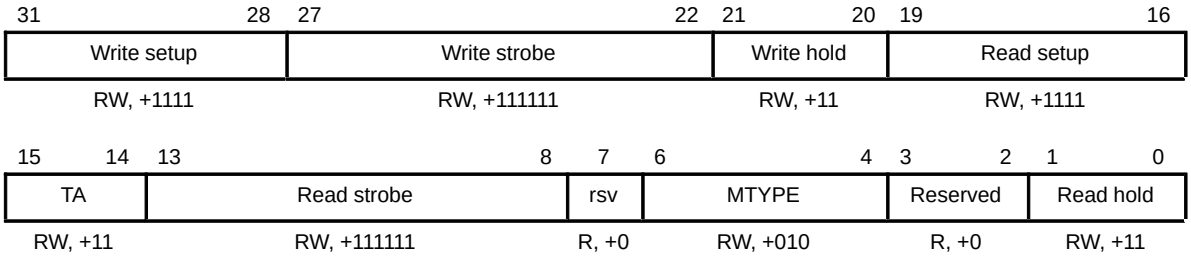
Table 6–3. EMIF Global Control Register Field Description

Field	Description
SDCINV	SDCLK polarity SDCINV = 0, SDCLK output is inverted from internal SDCLK SDCINV = 1, SDCLK output is identical to internal SDCLK
CLK2INV	CLKOUT2 polarity CLK2INV = 0, CLKOUT2 output is inverted from internal CLKOUT2 CLK2INV = 1, CLKOUT2 output is identical to internal CLKOUT2
ARDY	Value of ARDY input
$\overline{\text{HOLD}}$	Value of $\overline{\text{HOLD}}$ input
$\overline{\text{HOLDA}}$	Value of $\overline{\text{HOLDA}}$ output
NOHOLD	External HOLD disable (1 = hold disabled; 0 = hold enabled)
SDCEN	SDCLK enable SDCEN = 0, SDCLK held high SDCEN = 1, SDCLK enabled to clock
SSCEN	SSCLK enable SSCEN = 0, SSCLK held high SSCEN = 1, SSCLK enabled to clock
CLK1EN	CLKOUT1 enable CLK1EN = 0, CLKOUT1 held high CLK1EN = 1, CLKOUT1 enabled to clock
CLK2EN	CLKOUT2 enable CLK2EN = 0, CLKOUT2 held high CLK2EN = 1, CLKOUT2 enabled to clock
SSCRT	SBSRAM clock rate select SSCRT = 0, SSCLK 1/2x CPU clock rate SSCRT = 1, SSCLK 1x CPU clock rate
RBTR8	Requester arbitration mode RBTR8 = 0, requester controls EMIF until a high-priority request occurs RBTR8 = 1, requester controls EMIF for a minimum of eight accesses
MAP	Map mode, contains the value of the memory map mode of the device

6.2.2 CE Space Control Registers

The four CE space control registers (shown in Figure 6–4 and described in Table 6–4) correspond to the four CE memory spaces supported by the EMIF. The MTYPE field identifies the memory type for the corresponding CE space. If MTYPE selects SDRAM or SBSRAM, the remaining fields in the register do not have any effect. If an asynchronous type is selected (ROM or asynchronous), the remaining fields specify the shaping of the address and control signals for access to that space. Modification of a CE space control register does not occur until that CE space is inactive. These features are discussed in Section 6.5.

Figure 6–4. EMIF CE (0/1/2/3) Space Control Register Diagram



Note: R = Read
RW = Read/write

Table 6–4. EMIF Space Control Registers Field Descriptions

Field	Description
Read setup Write setup	Setup width. Number of CLKOUT1 cycles of setup time for address (EA) , chip enable ($\overline{CE}$), and byte enables ($\overline{BE}$[0-3]) before read strobe or write strobe falls. For asynchronous read accesses, this is also the setup time of $\overline{AOE}$ before $\overline{ARE}$ falls.
Read strobe Write strobe	Strobe width. The width of read strobe ($\overline{ARE}$) and write strobe ($\overline{AWE}$) in CLKOUT1 cycles.
Read hold Write hold	Hold width. Number of CLKOUT1 cycles that address (EA) and byte strobes ($\overline{BE}$[0-3]) are held after read strobe or write strobe rises. For asynchronous read accesses, this is also the hold time of $\overline{OE}$ after $\overline{ARE}$ rising.
MTYPE	Memory typeMTYPE = 000b, 8-bit-wide ROM MTYPE = 001b, 16-bit-wide ROM MTYPE = 010b, 32-bit-wide Asynchronous Interface MTYPE = 011b, 32-bit-wide SDRAM MTYPE = 100b, 32-bit-wide SBSRAM MTYPE = other, reserved

6.2.3 SDRAM Control Register

The SDRAM control register controls SDRAM parameters for all CE spaces that specify an SDRAM memory type in the MTYPE field of the associated CE space control register. Because the SDRAM control register controls all SDRAM spaces, each space must contain SDRAM with the same refresh and page characteristics. The fields in this register are discussed in Section 6.3.

Figure 6–5. EMIF SDRAM Control Register

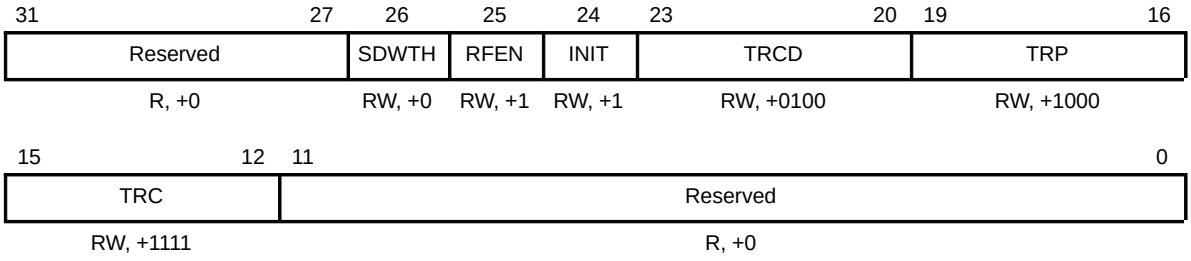


Table 6–5. EMIF SDRAM Control Register Field Description

Field	Description
TRC	Specifies t_{RC} value of the SDRAM in CLKOUT2 cycles. TRC = t_{RC} – 1
TRP	Specifies t_{RP} value of the SDRAM in CLKOUT2 cycles. TRP = t_{RP} – 1
TRCD	Specifies t_{RCD} value of the SDRAM in CLKOUT2 cycles. TRCD = t_{RCD} – 1
INIT	Forces initialization of all SDRAM present. See section 6.3.1. INIT = 0, no effect. INIT = 1, initialize SDRAM in each CE space configured for SDRAM.
RFEN	Refresh enable RFEN = 0, SDRAM refresh disabled. RFEN = 1, SDRAM refresh enabled.
SDWID	SDRAM width select SDWID = 0, Each external SDRAM space consists of four 8-bit SDRAMs SDWID = 1, Each external SDRAM space consists of two 16-bit SDRAMs

6.2.3.1 SDRAM Timing Register

The SDRAM refresh period register controls the refresh period in terms of CLKOUT2 cycles (1/2× the CPU clock rate). Optionally, the period field can send an interrupt to the CPU. Thus, this counter can be used as a general-purpose timer if SDRAM is not used by the system. The counter field can be read by the CPU. When the counter reaches 0, it is automatically reloaded with the period and an interrupt, (SDINT), is sent to the interrupt selector. See section 6.3.3 *Refresh* for more information on SDRAM refresh.

Figure 6–6. EMIF SDRAM Timing Register

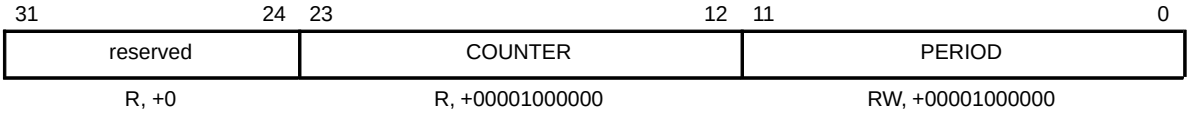


Table 6–6. EMIF SDRAM Timing Register Field Description

Field	Description
PERIOD	Refresh period in CLKOUT2 cycles
COUNTER	Current value of the refresh counter

6.3 SDRAM Interface

The EMIF supports 2 bank, 16M-bit and 4 bank, 64M-bit SDRAM, offering an interface to high-speed and high-density memory. The EMIF supports the SDRAM commands shown in Table 6–7. 16M-bit and 64M-bit SDRAM interfaces are shown in Figure 6–7 and Figure 6–8.

Table 6–9 describes the pin connection and related signals specific to SDRAM operation. Table 6–8 shows all of the possible SDRAM configurations available via the EMIF.

Table 6–7. EMIF SDRAM Commands

Command	Function
DCAB	Deactivate (also known as precharge) all banks
ACTV	Activate the selected bank and select the row
READ	Input the starting column address and begin the read operation
WRT	Input the starting column address and begin the write operation
MRS	Mode register set, configures SDRAM mode register
REFR	Autorefresh cycle with internal address

Figure 6–7. EMIF to 16M-Bit SDRAM Interface

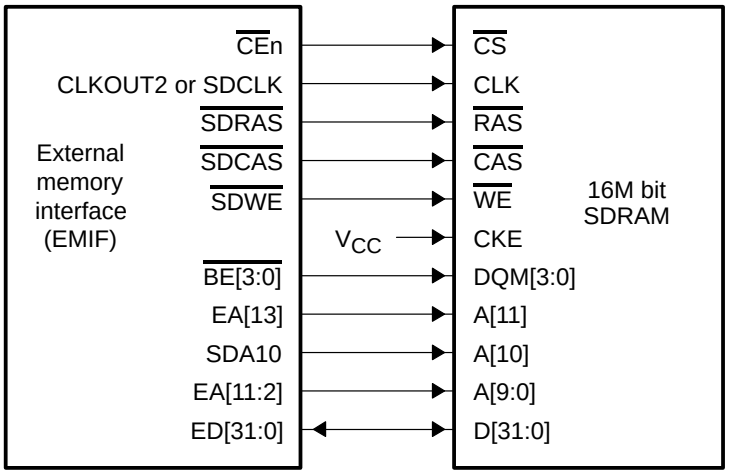


Figure 6–8. EMIF to 64M-Bit SDRAM Interface

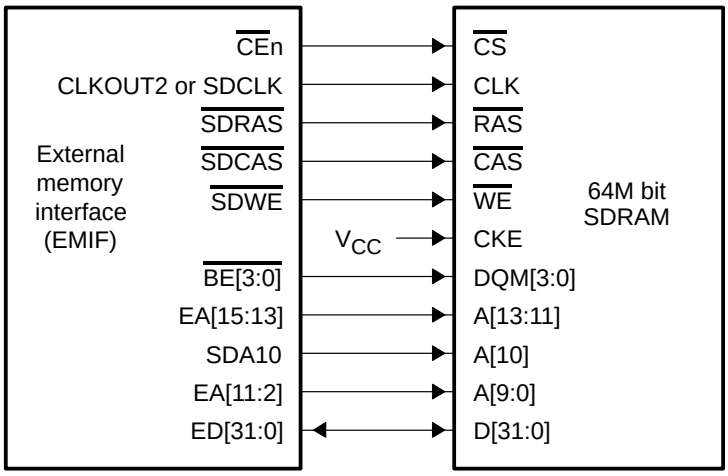


Table 6–8. SDRAM Memory Population

SDRAM Size	SDRAM Banks	SDRAM Width	Devices per CE Space	Memory Size per CE Space
16M-bit	2	16-bit	2	4M-bytes
16M-bit	2	8-bit	4	8M-bytes
64M-bit	4	16-bit	2	16M-bytes

Table 6–9. SDRAM Control Pins

EMIF Signal	SDRAM Signal	SDRAM Function
SDA10	A10	Address line A10/autoprecharge disable. Serves as a row address bit during ACTV commands and also disables the autoprecharging function of SDRAM.
$\overline{\text{SDRAS}}$	$\overline{\text{RAS}}$	Row address strobe and command input. Latched by the rising edge of CLK to determine current operation. Valid only if $\overline{\text{CS}}$ is active- low during that clock edge.
$\overline{\text{SDCAS}}$	$\overline{\text{CAS}}$	Column address strobe and command Input. Latched by the rising edge of CLK to determine current operation. Valid only if $\overline{\text{CS}}$ is active during that clock edge.
$\overline{\text{SDWE}}$	$\overline{\text{WE}}$	Write strobe and command input. Latched by the rising edge of CLK to determine current operation. Valid only if $\overline{\text{CS}}$ is active during that clock edge.
$\overline{\text{BE}}$[3:0]	DQM[3:0]	Data/output mask. DQM is an input/output buffer control signal. It disables writes and places outputs in the high impedance state during reads when high. DQM has a 2-CLK-cycle latency on reads and a 0-CLK-cycle latency on writes. DQM pins serve essentially as byte strobes and are connected to $\overline{\text{BE}}$[3:0] outputs.
$\overline{\text{CE}}$3 or $\overline{\text{CE}}$2 or $\overline{\text{CE}}$0	$\overline{\text{CS}}$	Chip select and command enable. $\overline{\text{CS}}$ must be active-low for a command to be clocked into the SDRAM. $\overline{\text{CS}}$ does not affect data input or output once a write or read has begun.
–	CKE	CKE clock enable. Tied high when interfaced to EMIF to enable clocking always.
CLKOUT2	CLK	SDRAM clock input. Runs at 1/2 the CPU clock rate.
SDCLK	CLK	SDRAM clock input. Runs at 1/2 the CPU clock rate. Equivalent to CLKOUT2.

6.3.1 SDRAM Initialization

The EMIF performs the necessary functions to initialize SDRAM if any of the CE spaces are configured for SDRAM. An SDRAM initialization is requested by a write of 1 to the INIT bit in the EMIF SDRAM control register.

The actual sequence of an initialization is as follows:

1) Send a DCAB command to all CE spaces configured as SDRAM.
2) Send three refresh commands.
3) Send an MRS command to all CE spaces configured as SDRAM.

The DCAB cycle is performed immediately after reset, provided the $\overline{\text{HOLD}}$ input is not active (a host request). If $\overline{\text{HOLD}}$ is active, the DCAB command is not performed until the hold condition is removed. The external requester should not attempt to access any SDRAM banks in this case, unless it performs SDRAM initialization and control itself.

6.3.2 Monitoring Page Boundaries

Because SDRAM is a paged memory type, the EMIF SDRAM controller monitors the active row of SDRAM so that row boundaries are not crossed during the course of an access. To accomplish this monitoring, the EMIF stores the address of the open page and performs compares against that address for subsequent accesses to the SDRAM bank. This storage and comparison is performed independently for each CE space.

The number of address bits compared is a function of the page size programmed in the SDWID field in the EMIF SDRAM control register. If SDWID = 0, the EMIF expects CE spaces configured as SDRAM to have four 8-bit wide SDRAMs that have page sizes of 512. Thus, the logical byte address bits compared are 25:11. If SDWID = 1, the EMIF expects CE spaces with SDRAM to have two 16-bit-wide SDRAMs that have page sizes of 256. Thus, the logical byte address bits compared are 24:10.

If, during the course of an access, a page boundary is crossed, the EMIF performs a DCAB command and starts a new row access. Also, a change in direction of an access (read to write, or write to read) causes a page miss.

Simply ending the current access is not a condition that forces the active SDRAM row to be closed. The EMIF leaves the active row open until it becomes necessary to close it. This feature decreases the deactivate-reactivate overhead and allows the interface to capitalize fully on address locality of memory accesses.

6.3.3 Refresh

The RFEN bit in the SDRAM control register selects the SDRAM refresh mode of the EMIF. A value of 0 in RFEN disables all EMIF refreshes, and you must ensure that refreshes are implemented in an external device. A value of 1 in RFEN enables the EMIF to perform refreshes of SDRAM.

Refresh commands (REFR) enable all $\overline{CE}$ signals for all CE spaces selected to use SDRAM (with the MTYPE field of the CE space control register). REFR is automatically preceded by a DCAB command. This ensures all CE spaces selected with SDRAM are deactivated. Following the DCAB command, the EMIF begins performing trickle refreshes at a rate defined by the period value in the EMIF SDRAM control register, provided no other SDRAM access is pending.

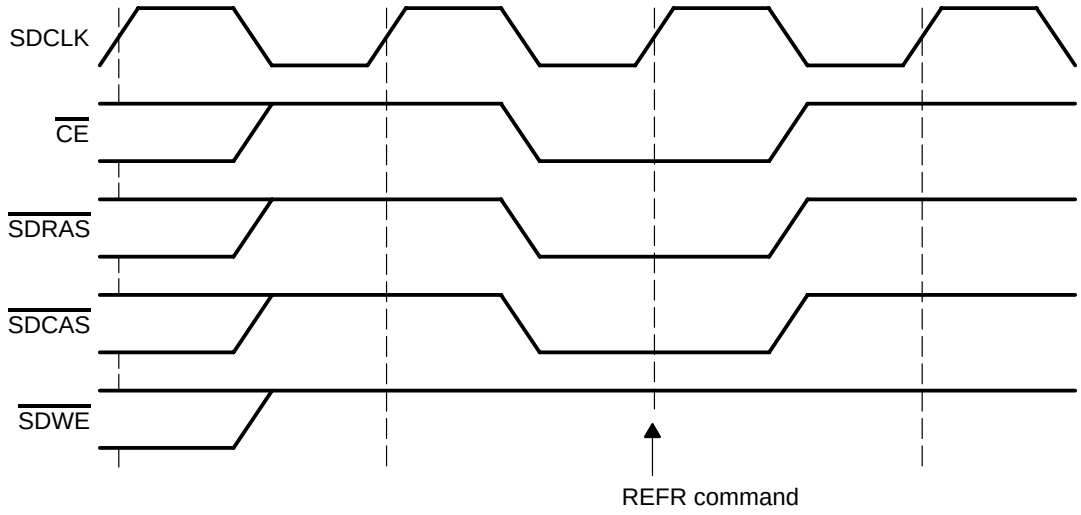
The SDRAM interface monitors the number of refresh requests posted to it and performs the refreshes. Within the EMIF SDRAM control block, a 2-bit counter monitors the backlog of refresh requests. The counter increments once for each refresh request and decrements once for each refresh cycle performed. The counter saturates at the value of 11, and also at 00. At reset, the counter is automatically set to 11 to ensure that several refreshes occur before accesses begin.

The EMIF SDRAM controller prioritizes SDRAM refresh requests with other data access requests posted to it from the EMIF requesters. The following rules apply:

- A counter value of 11 invalidates the page information register, forcing the controller to close the current SDRAM page. The value of 11 indicates an urgent refresh condition. Thus, the EMIF SDRAM controller performs three REFR commands, thereby decrementing the counter to 00 following the DCAB command before proceeding with the remainder of the current access. If SDRAM is present in multiple CE spaces, the DCAB-refresh sequence occurs in all spaces containing SDRAM.
- During idle times on the SDRAM interface(s), if no request is pending from the EMIF, the SDRAM interface performs REFR commands as long as the counter value is nonzero. This feature reduces the likelihood of having to perform urgent refreshes during actual SDRAM accesses later. If SDRAM is present in multiple CE spaces, this refresh occurs only if all interfaces are idle with invalid page information.

The EMIF SDRAM interface performs $\overline{\text{CAS}}$-before-$\overline{\text{RAS}}$ refresh cycles for SDRAM. Some SDRAM manufacturers call this autorefresh. Prior to a REFR command, a DCAB command is performed to all CE spaces specifying SDRAM to ensure that all active banks are closed. Page information is always invalid before and after a REFR command; thus, a refresh cycle always forces a page miss. A deactivate cycle is required prior to the refresh command.

Figure 6–9. SDRAM Refresh



6.3.4 Mode Register Set

The EMIF automatically performs a DCAB command followed by an MRS command whenever the INIT field in the EMIF SDRAM control register is set. INIT can be set by device reset or by a user write. Like DCAB and REFR commands, MRS commands are performed to all CE spaces configured as SDRAM through the MTYPE field. Following the MRS cycle, the INIT bit clears itself to prevent multiple MRS cycles. Following a hold, the external requester should return it to its original value before returning control of the bus to the EMIF. Alternatively, you could poll the HOLD and HOLDA bits in the EMIF global control register and, upon detecting completion of an external hold, reinitialize the EMIF by setting the INIT bit in the EMIF SDRAM control register.

The EMIF always uses a mode register value of 0x0030 during a MRS command. Figure 6–10 shows the mapping between mode register bits, EMIF pins, and the mode register value.

Table 6–10 shows the SDRAM configuration selected by this mode register value.

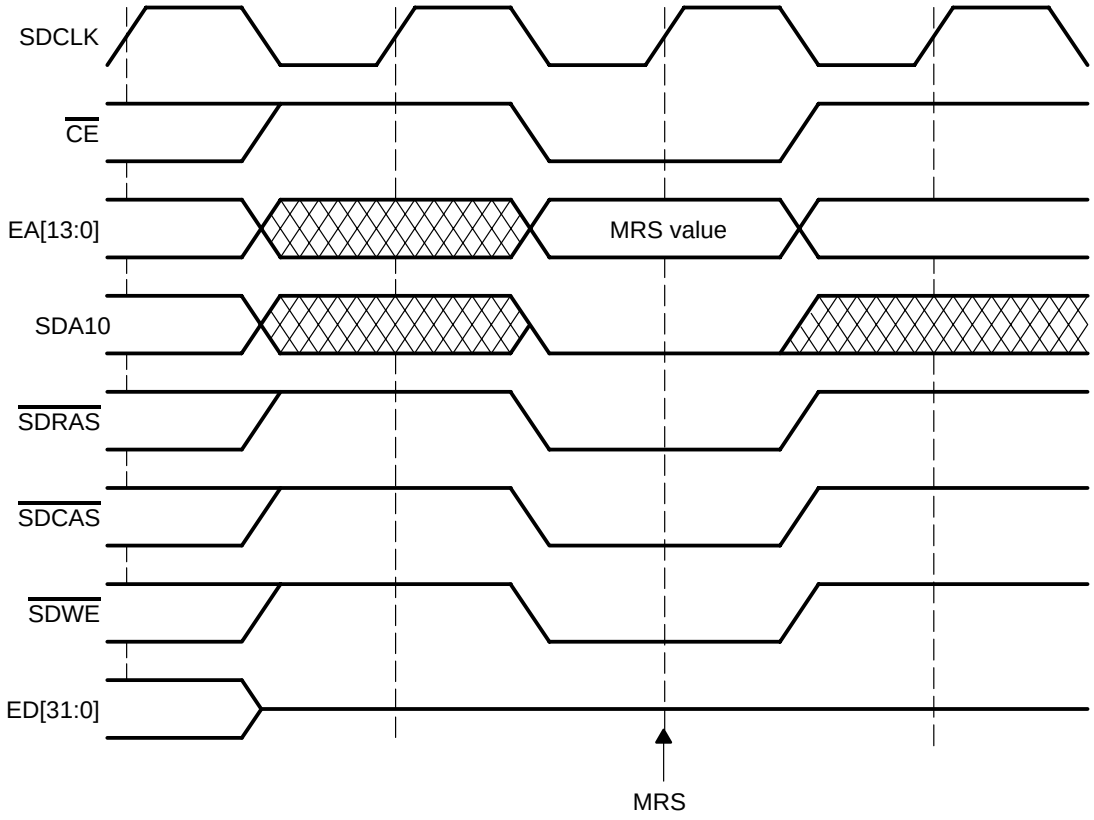
Figure 6–10. Mode Register Value

Mode register bit	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EMIF Pins	EA15	EA14	EA13	SDA10	EA11	EA10	EA9	EA8	EA7	EA6	EA5	EA4	EA3	EA2
Field	Reserved				Write burst length	Reserved		Read latency			S/I	Burst length		
Value	0	0	0	0	0	0	0	0	1	1	0	0	0	0

Table 6–10. Implied SDRAM Configuration by MRS Value

Field	Selection
Write burst length	1
Read latency	3
Serial/interleave burst type	Serial
Burst length	1

Figure 6–11. SDRAM Module Register Set: MRS Command



6.3.5 Address Shift

Because the same EMIF pins address the row and column address, the EMIF interface appropriately shifts the address in row and column address selection. Table 6–11 shows the translation between bits of the byte address and how they appear on the EA pins for row and column addresses. SDRAMs use the address inputs for control as well as address. With this consideration, the following items clarify the figure:

- The address line that corresponds to the SDRAM's bank select field (EA11 on 16M-bit SDRAM; EA13 and EA12 on 64M-bit SDRAM) is latched internally by the SDRAM controller. This ensures that the bank select remains correct during READ and WRT commands. Thus, the EMIF maintains these values as shown in both row and column addresses.
- The EMIF forces the address bit below bank select (SDA10) to be low unless $\overline{RAS}$ is active, yet high during DCAB commands at the end of a page of accesses. This prevents the autoprecharge from occurring following a READ or WRT command.

Table 6–11. Byte Address to EA Mapping for SDRAM $\overline{RAS}$ and $\overline{CAS}$

SDRAM width	SDWID	DRAM command	EA22–EA17	EA16	EA15	EA14	EA13	SDA10	EA11	EA10	EA9	EA8	EA7	EA6	EA5	EA4	EA3	EA2
x16	1	RAS		24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
		CAS		24	23	22	21	20	19	18	9	8	7	6	5	4	3	2
x8	0	RAS		24	23	22	21	20	19	18	17	16	15	14	13	12	11	
		CAS		24	23	22	21	20	10	9	8	7	6	5	4	3	2	

Legend:
- Bit is internally latched during ACTV command
- Reserved for future use. Undefined.

Note: The $\overline{RAS}$ and $\overline{CAS}$ values indicate the bit of the byte address present on the corresponding EA pin during a $\overline{RAS}$ or $\overline{CAS}$ cycle.

6.3.6 Timing Requirements

Five SDRAM timing parameters decouple the EMIF from SDRAM speed limitations. Three of these parameters are programmable via the EMIF SDRAM control register; the remaining two are assumed to be static values as shown in Table 6–12. The three programmable values assure that EMIF control of SDRAM obeys these minimum timing requirements. Consult the SDRAM data sheet for the parameters appropriate for your particular SDRAM.

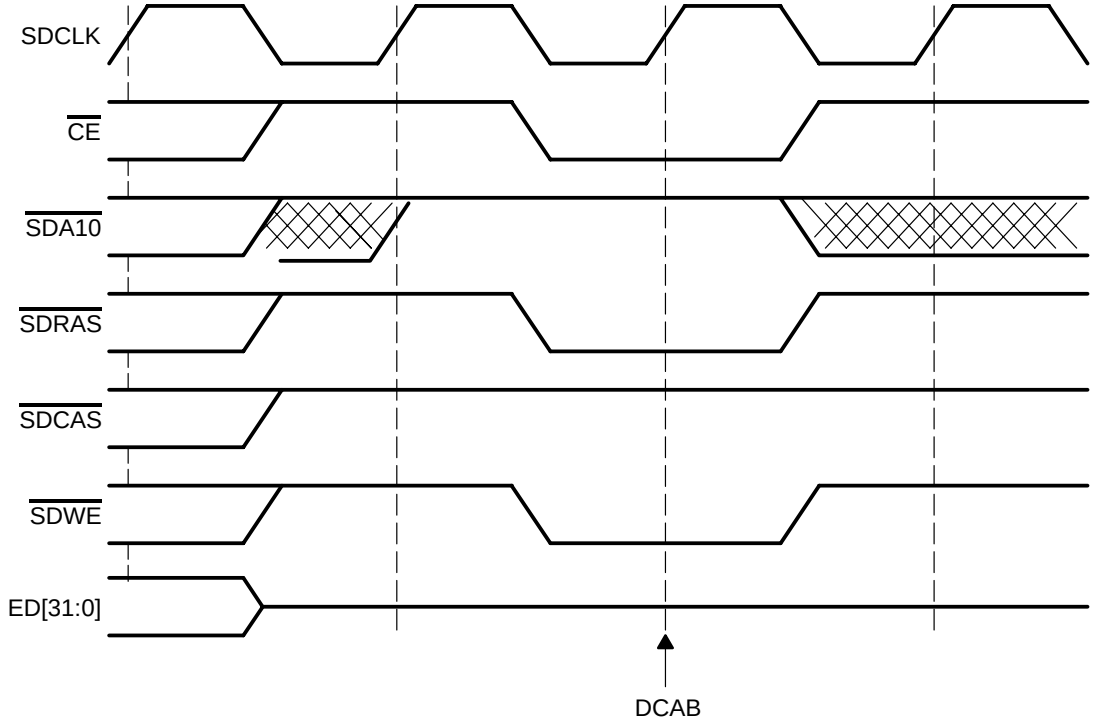
Table 6–12. SDRAM Timing Parameters

Parameter	Description	Value in CLKOUT2 Cycles
t_{RC}	REFR command to ACTV, MRS, or subsequent REFR command	TRC + 1
t_{RCD}	ACTV command to READ or WRT command	TRCD + 1
t_{RP}	DCAB command to ACTV, MRS, or REFR command	TRP +1
t_{RAS}	ACTV command to DEAC to DCAB command	7
t_{nEP}	Overlap between read data and a DCAB command	2

6.3.7 Deactivation

The SDRAM deactivation (DCAB) is performed after a hardware reset or when INIT = 1 in the EMIF SDRAM control register. This cycle is also required by the SDRAMs prior to REFR, MRS, and when a page boundary is crossed. During the DCAB command, SDA10 is driven high to ensure that all SDRAM banks are deactivated.

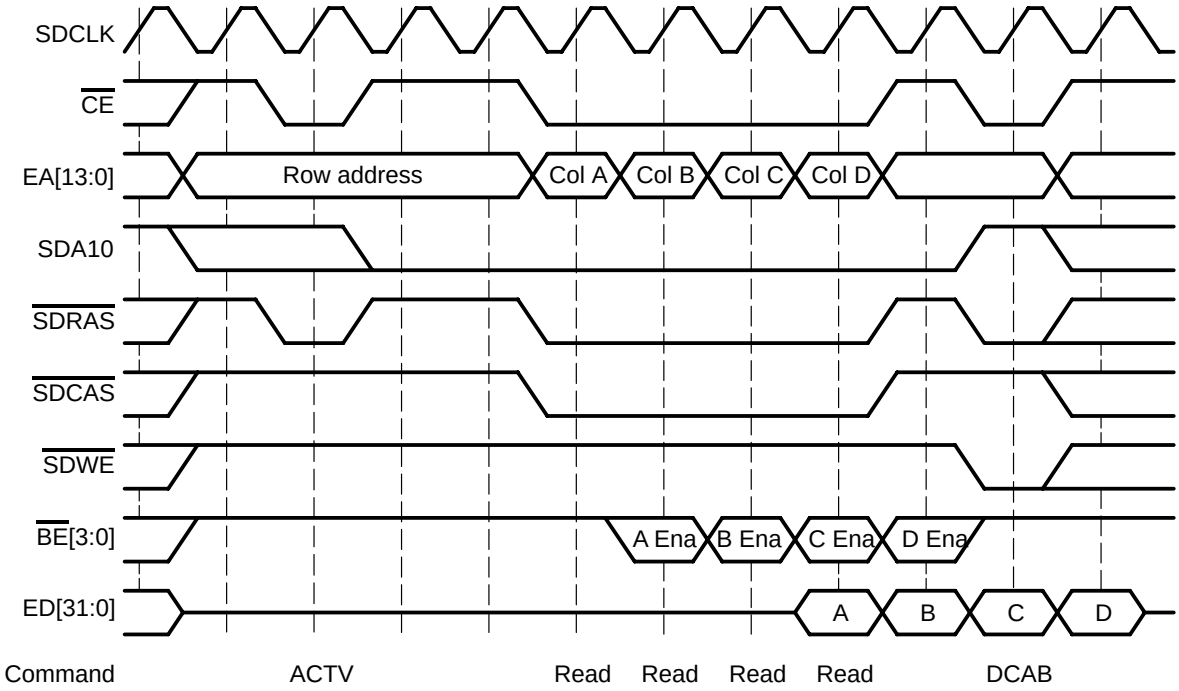
Figure 6–12. SDRAM Deactivation



6.3.8 SDRAM Read

During an SDRAM read, the selected bank is activated with the row address during the ACTV command. In this example, four read commands are performed at four different column addresses. The EMIF uses a $\overline{CAS}$ latency of 3 and a burst length of 1. The 3-cycle latency causes data to appear three cycles after the corresponding column address. Following the last column access, a DCAB cycle is performed to deactivate the bank. An idle cycle is inserted between the final read command and the DCAB command to meet SDRAM timing requirements. The transfer of data finishes during and past the DCAB command. If no new access is pending, the DCAB command is not performed until such time as the page information becomes invalid (see section 6.3.2). The values on EA[13:11] during column accesses and the DCAB command are the values latched during the ACTV command.

Figure 6–13. SDRAM Read With $\overline{CAS}$ Latency 3, Burst Length 1



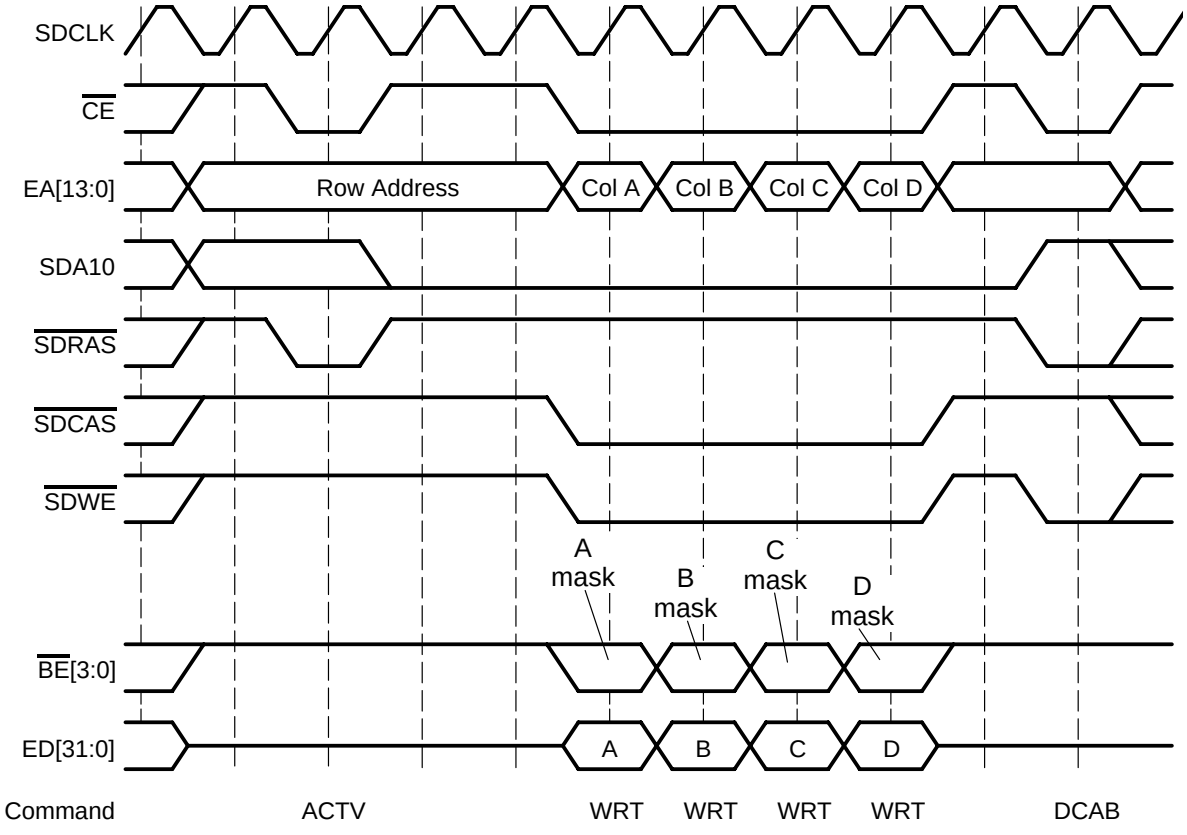
Note:

Cycle shown has t_{RCD} = 10 (decimal 2).

6.3.9 SDRAM Write

All SDRAM writes have a burst length of 1. The bank is activated with the row address during the ACTV command. There is no latency on writes, so data is output on the same cycle as the column address. Writes to particular bytes are disabled via the appropriate DQM inputs; this feature allows for byte and halfword writes. Following the final write command, an idle cycle is inserted to meet SDRAM timing requirements. The bank is then deactivated with a DCAB command and the memory interface can begin a new page access. If no new access is pending, the DCAB command is not performed until such time as the page information becomes invalid (see section 6.3.2). The values on EA[13:11] during column accesses and the DEAC command are the values latched during the ACTV command.

Figure 6–14. SDRAM Write With Burst Length 1



Note:

Cycle shown has t_{RCD} = 10b (decimal 2).

6.4 SBSRAM Interface

As shown in Figure 6–15, the EMIF interfaces directly to industry-standard synchronous burst SRAMs. This memory interface allows a high-speed memory interface without some of the limitations of SDRAM. Most notably, since SBSRAM are SRAM devices, random accesses in the same direction can occur in a single cycle. The SBSRAM interface can run at either the CPU clock speed or at 1/2 of this rate. The selection is made based on the setting of the SSCRT bit in the EMIF global control register.

The four SBSRAM control pins are latched by the SBSRAM on the rising SSCLK edge to determine the current operation. These signals are valid only if the chip select line for the SBSRAM is low. The $\overline{\text{ADV}}$ signal is used to allow the SBSRAM device to generate addresses internally for interfacing to controllers that cannot provide addresses quickly enough. The EMIF does not need to use this signal because it can generate the address at the required rate.

Figure 6–15. EMIF-SBSRAM Interface

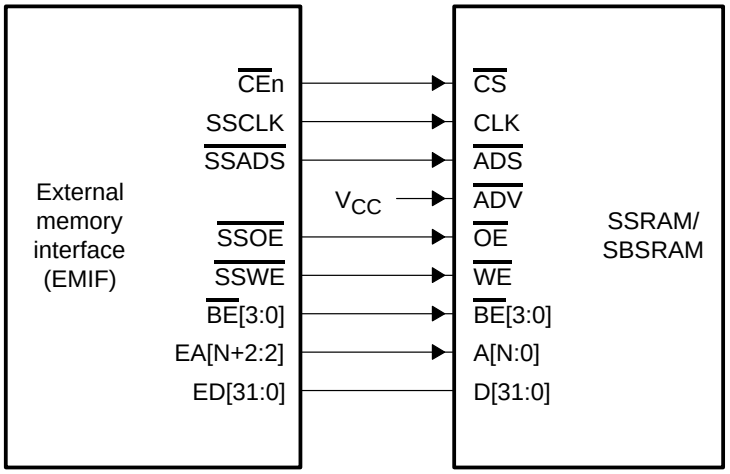


Table 6–13. EMIF SBSRAM Pins

EMIF Signal	SBSRAM Signal	SBSRAM Function
$\overline{\text{SSADS}}$	$\overline{\text{ADS}}$	Address strobe
$\overline{\text{SSOE}}$	$\overline{\text{OE}}$	Output enable
$\overline{\text{SSWE}}$	$\overline{\text{WE}}$	Write enable
SSCLK	CLK	SBSRAM clock

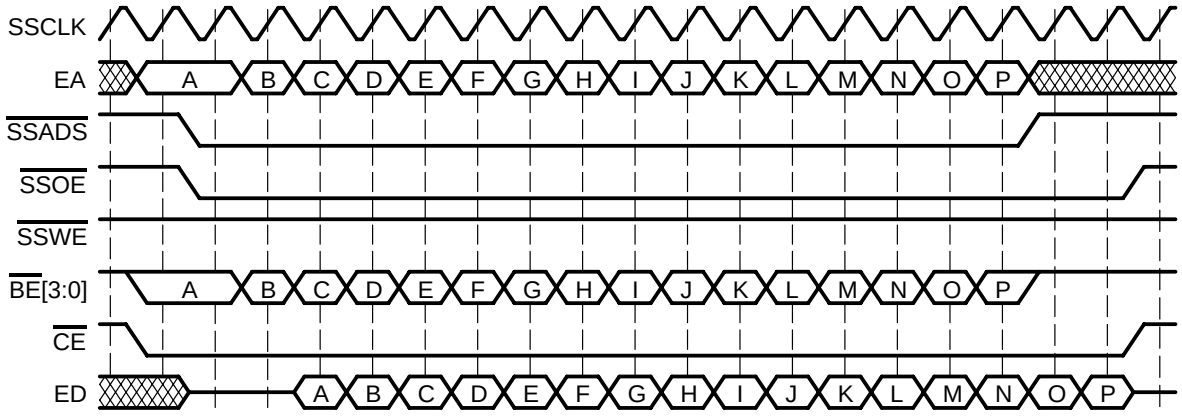
6.4.1 Optimizing SBSRAM Accesses

SBSRAMs are latent by their architecture, meaning that read data follows address and control information. Consequently, the EMIF inserts cycles between read and write commands to ensure that no conflict exists on the ED[31:0] bus. The EMIF keeps this turnaround penalty to a minimum. The initial 2-cycle penalty is present when changing directions on the bus. In general, the first access of a burst sequence incurs a 2-cycle startup penalty.

6.4.2 SBSRAM Reads

Figure 6–16 shows a 16-word read of an SBSRAM. Every access strobes a new address into the SBSRAM, indicated by the $\overline{\text{SSADS}}$ strobe low. The first access requires an initial startup penalty of two cycles; thereafter, all accesses occur in a single SSCLK cycle.

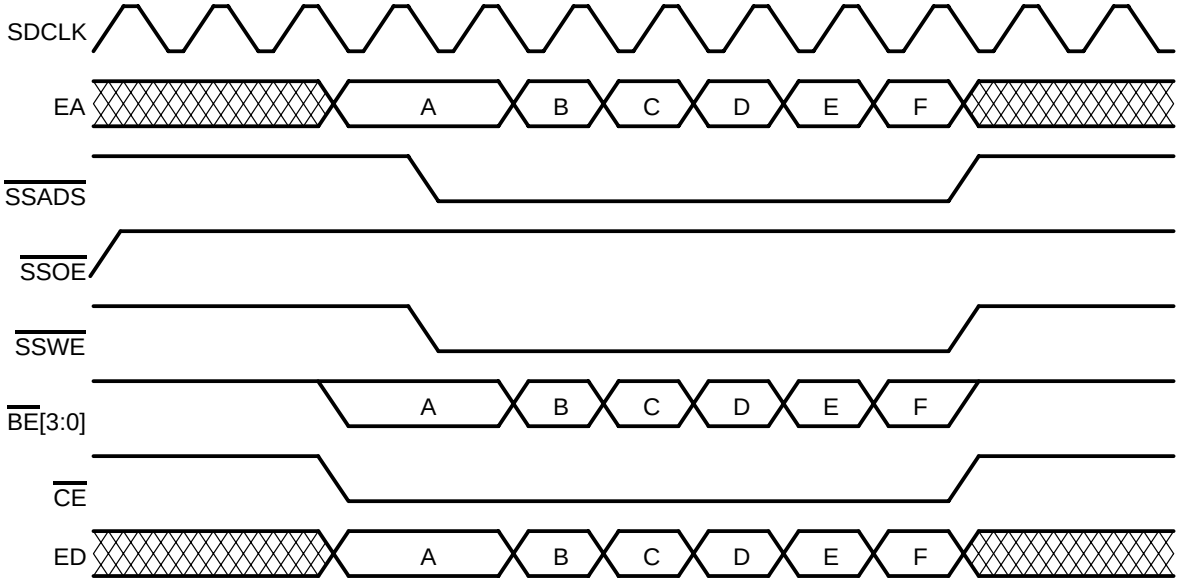
Figure 6–16. SBSRAM Read of 16 Words



6.4.3 SBSRAM Writes

Figure 6–17 shows a 6-word write of an SBSRAM. Every access strobes a new address into the SBSRAM. The first access requires an initial startup penalty of two cycles; thereafter, all access can occur in a single SSCLK cycle.

Figure 6–17. SBSRAM Write of Six Words



6.5 Asychronous Interface

The asynchronous interface offers configurable memory cycle types, to interface to a variety of memory and peripheral types, including SRAM, EPROM, Flash memory, as well as FPGA and ASIC designs.

Table 6–14 lists the asynchronous interface pins.

Figure 6–18 shows an interface to standard SRAM. Figure 6–19 shows an interface to a FIFO buffer. Figure 6–20, Figure 6–21, and Figure 6–22 show interfaces to 8-, 16-, and 32-bit ROM. Although ROM can be interfaced at any of the CE spaces, it is often used at CE1 because that space can be configured for widths of less than 32 bits.

Table 6–14. EMIF Asychronous Interface Pins

EMIF Signal	Function
$\overline{\text{AOE}}$	Output enable. Active-low during the entire period of a read access.
$\overline{\text{AWE}}$	Write enable. Active-low during a write transfer strobe period.
$\overline{\text{ARE}}$	Read enable. Active-low during a read transfer strobe period.
$\overline{\text{ARDY}}$	Ready. Input used to insert wait states into the memory cycle.

Figure 6–18. EMIF SRAM Interface

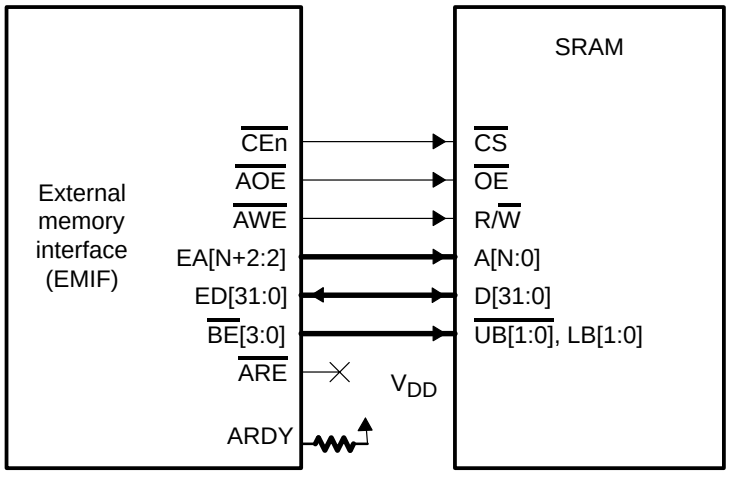


Figure 6–19. EMIF-FIFO Interface

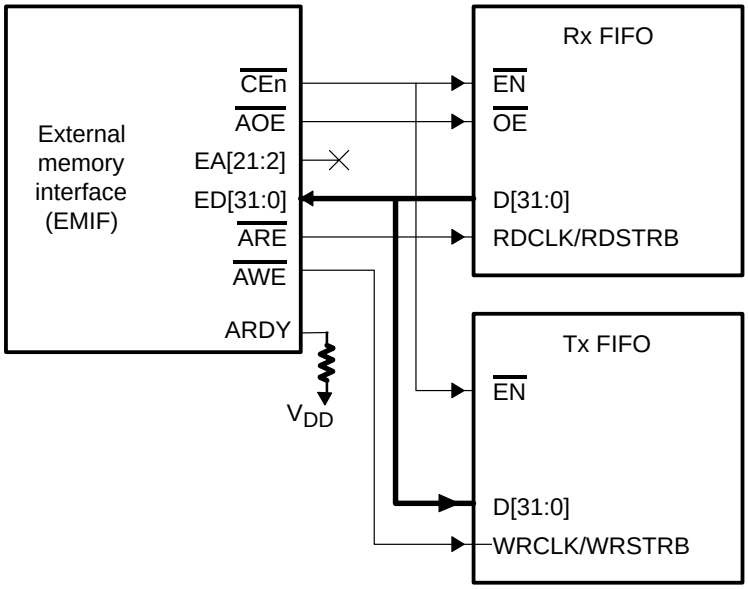


Figure 6–20. EMIF-ROM 8-Bit Interface

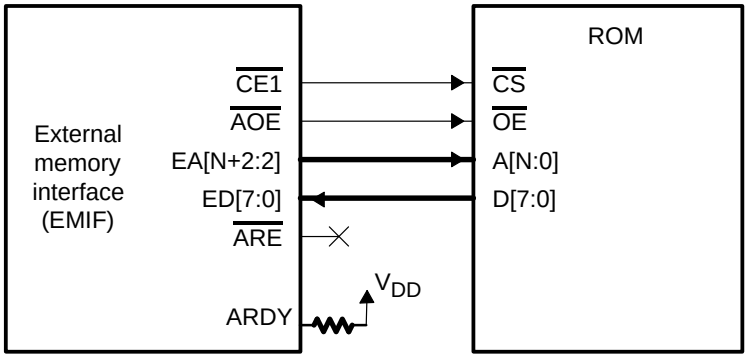


Figure 6–21. EMIF-ROM 16-Bit Interface

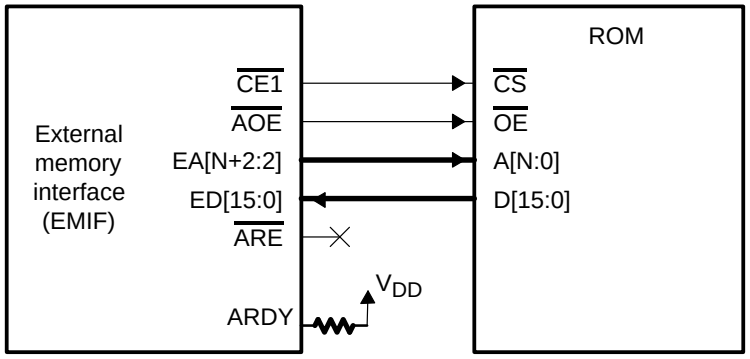
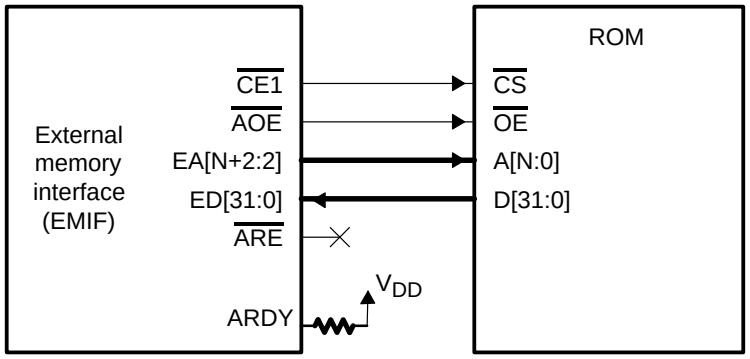


Figure 6–22. EMIF-ROM 32-Bit Interface



6.5.1 ROM Modes

The EMIF supports 8- and 16-bit wide ROM access modes as selected by the MTYPE field in the EMIF CE space control register. In reading data from these narrow-width memory spaces, the EMIF packs multiple reads into one 32-bit-wide value. This mode is primarily intended for word access to 8-bit and 16-bit ROM devices. Thus, the following restrictions apply:

- Read operations always read 32 bits, regardless of the access size or the memory width.
- The address is shifted up appropriately to provide the correct address to the narrow memory. The shift amount is 1 for 16-bit ROM and 2 for 8-bit ROM. Thus, the high address bits are shifted out and accesses wrap around if that $\overline{CE}$ space spans the entire EA bus. Table 6–15 shows the address bits present on the EA bus during an access to CE1 space for all possible asynchronous memory widths.
- The EMIF always reads the lower addresses first and packs these into the LSBytes and packs subsequent accesses into the higher order bytes. Thus, the expected packing format in ROM is always little-endian, regardless of the value of the LENDIAN bit.

Table 6–15. Byte Address to EA Mapping for Asynchronous Memory Widths

	EA Line																				
Width	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
×32	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
×16	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
×8	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

6.5.1.1 8-Bit ROM

In 8-bit ROM mode, the address is left shifted by 2 to create a byte address on EA to access byte-wide ROM. The EMIF always packs four consecutive bytes aligned on a 4-byte (byte address = 4N) boundary into a word access. The bytes are fetched in the following address order: 4N, 4N + 1, 4N + 2, 4N + 3. Bytes are packed into the 32-bit word in the following little endian order from MSByte to LSByte: 4N + 3, 4N + 2, 4N + 1, 4N.

6.5.1.2 16-Bit ROM

In 16-bit ROM mode, the address is left shifted by 1 to create a half-word address on EA to access 16-bit-wide ROM. The EMIF always packs two consecutive half-words aligned on a 4-byte (byte address = 4N) boundary into a word access. The halfwords are fetched in the following address order: 4N, 4N + 2. Halfwords are packed into the 32-bit word in the following little-endian order from MSHalfword to LSHalfword: 4N + 2, 4N.

6.5.2 Programmable ASRAM Parameters

The EMIF allows a high degree of programmability for shaping asynchronous accesses. The programmable parameters that allow this are:

- **Setup:** The time between the beginning of a memory cycle ($\overline{CE}$ low, address valid) and the activation of the read or write strobe
- **Strobe:** The time between the activation and deactivation of the read ($\overline{ARE}$) or write strobe ($\overline{AWE}$)
- **Hold:** The time between the deactivation of the read or write strobe and the end of the cycle (which can be either an address change or the deactivation of the $\overline{CE}$ signal)

These parameters are programmable in terms of CPU clock cycles via fields in the EMIF CE space control registers. Separate setup, strobe, and hold parameters are available for read and write accesses. The setup and strobe fields have minimum count of 1. For setup and strobe, a count of 0 is treated as a count of 1. Hold can be set to 0 cycles. Figure 6–23 illustrates these parameters.

6.5.3 Asynchronous Reads

Figure 6–23 illustrates an asynchronous read with SETUP and HOLD set to 1 and STROBE set to 3. An asynchronous read proceeds as follows:

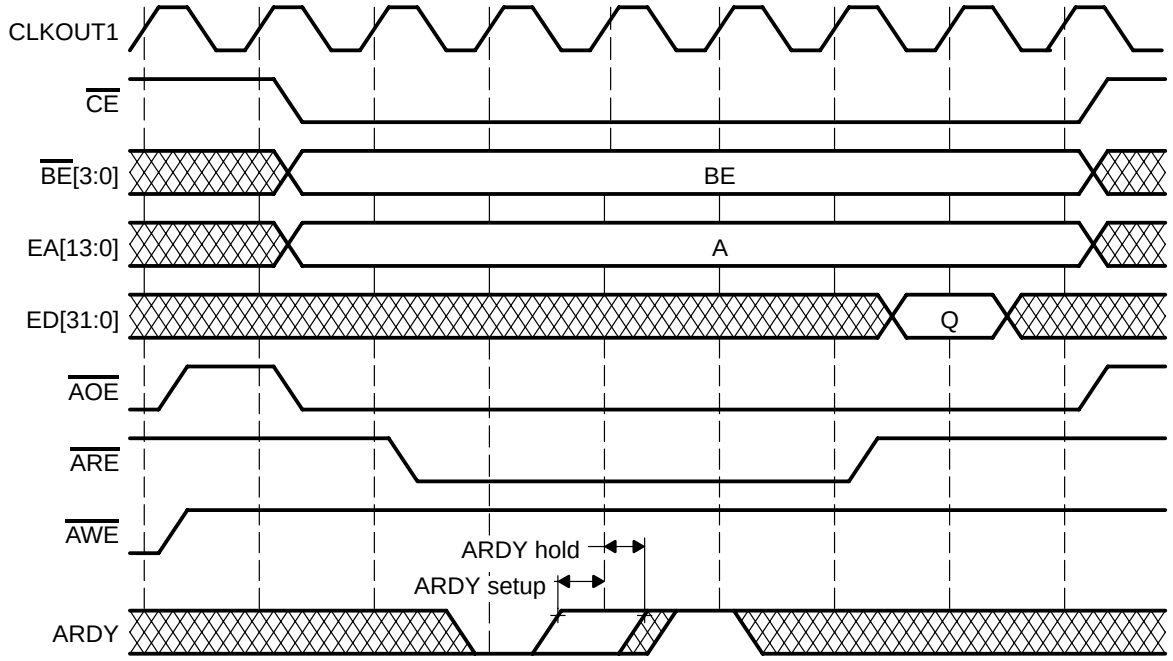
- At the beginning of the setup period:
  - $\overline{CE}$ becomes active
  - $\overline{AOE}$ becomes active
  - $\overline{BE}$[3:0] becomes valid.
  - EA becomes valid.
- At the beginning of a strobe period, $\overline{ARE}$ becomes active
- At the beginning of a hold period:

- $\overline{ARE}$ becomes inactive high
- Data is sampled on the CLKOUT1 rising edge concurrent with the beginning of the hold period (end of the strobe period) and just prior to the $\overline{ARE}$ low-to-high transition.

❑ At the end of the hold period:

- $\overline{CE}$ becomes inactive as long as another read access to the same $\overline{CE}$ space is not scheduled for the next cycle.
- $\overline{AOE}$ becomes inactive as long as another read access to the same $\overline{CE}$ space is not scheduled for the next cycle.

Figure 6–23. Asynchronous Read Timing Example



6.5.4 Asynchronous Writes

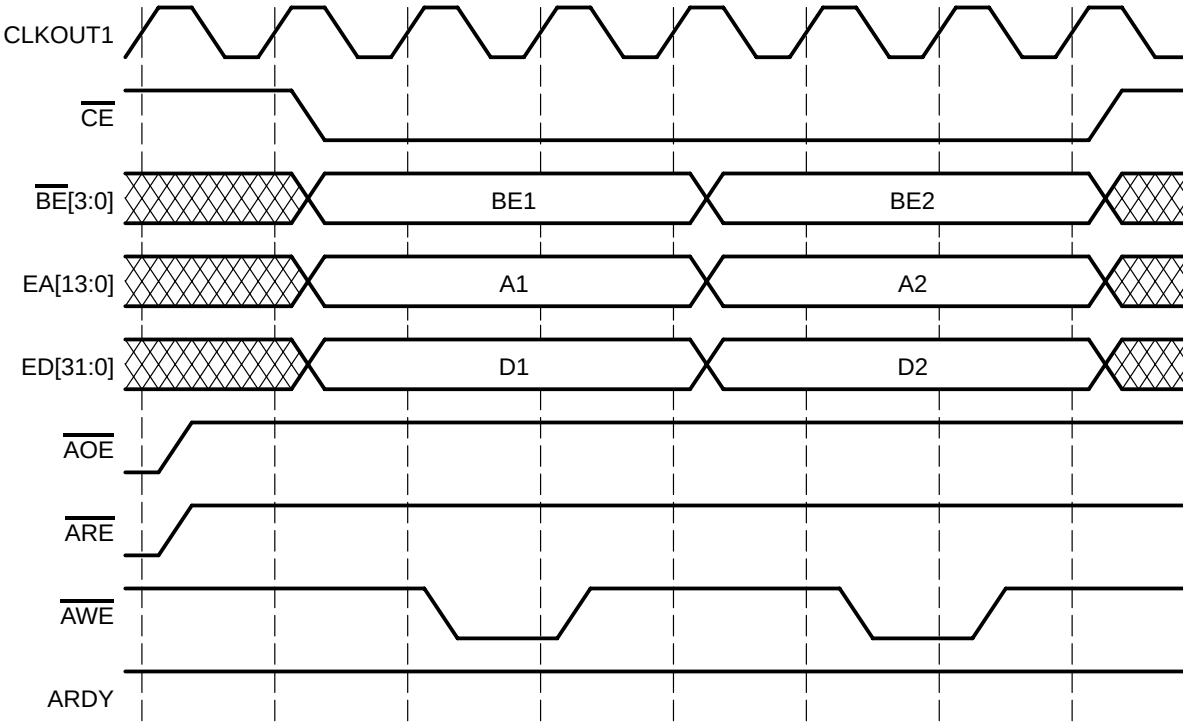
Figure 6–24 depicts two back-to-back asynchronous write cycles with the ARDY signal pulled high (always ready).

❑ At the beginning of the setup period:

- $\overline{CE}$ becomes active
- $\overline{BE}$[3:0] becomes valid.
- EA becomes valid.
- ED becomes valid.

- At the beginning of a strobe period, $\overline{AWE}$ becomes inactive.
- At the beginning of a hold period:
  - $\overline{AWE}$ becomes inactive
  - Data is sampled on the CLKOUT1 rising edge concurrent with the beginning of the hold period (end of the strobe period) and just prior to the $\overline{AWE}$ low-to-high transition.
- At the end of the hold period.
  - ED goes into the high-impedance state only if as another write access to the same /CE space is NOT scheduled for the next cycle.
  - $\overline{CE}$ becomes inactive only if another write access to the same $\overline{CE}$ space is not scheduled for the next cycle.
  - If the next access is a read a cycle of turn around is inserted before $\overline{AOE}$ is enabled.

Figure 6–24. Asynchronous Write Timing Example



6.5.5 Ready Input

The read cycle in Figure 6–23 illustrates read operation. In addition to programmable access shaping, you can insert extra cycles into the strobe period

by deactivating the ARDY input. The ready input is internally synchronized to the CPU clock. Ready operation is as follows:

- If during a rising edge of CLKTOUT1 the STROBE count is 1 or 0, and the ARDY signal was active during the previous rising edge, the next e\second rising edge of CLKOUT1 will end the ARE_ or AWE_ strobe. Otherwise the read or write cycle will be extended by at least one more CLKOUT1 period.

- The miniumum strobe field width value is 3 if inactive (low). ARDY is used to extend the asynchronous read or write cycles. Otherwise, if the ARDY is always active (high), the minimum strobe width is 1.

6.6 Hold Interface

The EMIF responds to hold requests for the external bus. The hold handshake allows an external device and the EMIF to share the external bus. The handshake mechanism uses two signals:

- $\overline{\text{HOLD}}$: Asynchronous hold request input. The external device drives this pin low to request bus access. $\overline{\text{HOLD}}$ is the highest priority request that the EMIF can receive during active operation. When the hold is requested, the EMIF stops driving the bus at the earliest possible moment, which may entail completion of the current accesses, device deactivation, and SDRAM bank deactivation. The external device must continue to drive $\overline{\text{HOLD}}$ low for as long as it wants to drive the bus. The $\overline{\text{HOLD}}$ input is internally synchronized to the CPU clock.

- $\overline{\text{HOLDA}}$: Hold acknowledge output. The EMIF asserts this signal active after it has placed its signal outputs in the high-impedance state. The external device may then drive the bus as required. The EMIF places all outputs in the high-impedance state with the exception of the clock outputs: CLKOUT1, CLKOUT2, SDCLK, and SSCLK.

There is no mechanism to ensure that the external device does not attempt to drive the bus indefinitely. The user should be aware of system-level issues, such as refresh, that may need to be performed. During host requests, the refresh counters within the EMIF continue to log refresh requests; however, no refresh cycles can be performed until bus control is granted back to the EMIF by returning the $\overline{\text{HOLD}}$ input to the inactive level. You can prevent an external hold by setting the NOHOLD bit in the EMIF global control register.

6.7 Priority

Table 6–16 illustrate the priority scheme that the EMIF uses in the case of multiple pending requests. The priority scheme may change if the DMA channel that is issuing a request through the DMA controller is of high priority. This mode is set in the DMA controller by setting the PRI bit in the DMA channel primary control register.

Once a requester (in this instance, the refresh controller is considered a requester) is prioritized and chosen, no new requests are recognized until either the chosen requester stops making requests or a subsequent higher priority request occurs. In this case, all issued requests of the previous requester are allowed to finish while the new requester starts making its requests.

If the arbitrate bit of the EMIF global control register is set (RBTR8 = 1), once a requester gains control of the EMIF it maintains control as long as the requester needs the EMIF; or if eight word requests have occurred, until a higher priority requester requests the EMIF. If a higher priority requester needs the EMIF, it will not get control until the current controller relinquishes control or until eight word requests have finished. If the arbitrate bit is not set (RBTR8 = 0), a requester maintains control of the EMIF as long as it needs the EMIF or until a higher priority requester requests the EMIF. The current controller is interrupted by a higher priority requester, regardless of the number of requests that have occurred.

Table 6–16. EMIF Prioritization of Requests

Priority	Requester PRI = 1	Requester PRI = 0
Highest	External hold	
	Mode register set	
	Urgent refresh	
	DMA controller	DMC
	DMC	PMC
	PMC	DMA controller
Lowest	Trickle refresh	

6.8 Clock Output Enabling

To reduce EMI radiation, the EMIF allows disabling (holding high) CLKOUT2, CLKOUT1, SSCLK, and SDCLK. This feature is performed via the CLK2EN, CLK1EN, SSCEN, and SDCEN bits in the EMIF SDRAM control register.

6.9 Emulation Halt Operation

The EMIF continues operating during emulation halts. Emulator accesses through the EMIF can have the effect of changing EMIF state and forcing startup penalties once the halt is stopped.

6.10 Power Down

In power down 2 mode, refresh is enabled

SSCLK, CLKOUT1, CLKOUT2 are held low during powerdown 2 and power-down 3 modes.

In power down 3 mode, EMIF acts as if it were in reset.

Chapter 7

Boot Configuration, Reset, and Memory Map

Describes the boot modes and associated memory maps available for the TMS320C32062xx.

Topic **Page**

7.1 Overview

The ’C62xx uses a variety of boot configurations to determine what actions the device is to perform after reset for proper device initialization. The configurable options include:

- Selection of the memory map which determines whether internal or external memory is mapped at address 0
- Selection of the type of external memory is mapped at address 0 if external memory is indeed mapped there
- Selection of the boot process used to initialize the memory at address 0 before the CPU is allowed to run

7.2 Device Reset

The external device reset uses an active-low signal, $\overline{\text{RESET}}$. While $\overline{\text{RESET}}$ is low, the device is held in reset and is initialized to the prescribed reset state. All 3-state outputs are placed in a state of high impedance, and all other outputs are returned to their default states. $\overline{\text{RESET}}$ is latched with the device clock input (CLKIN) as well as the CPU clock (CLKOUT1). Thus, $\overline{\text{RESET}}$ has a minimum low time in terms of CLKIN as well as CPU clock cycles. The precise timing requirements for device reset are described in the specific device data sheet. The rising edge of $\overline{\text{RESET}}$ starts the processor running with the prescribed boot configuration. The length of the $\overline{\text{RESET}}$ pulses may have to be stretched if the PLL needs to sync up followinug power-up or when PLL configuration pins change during reset.

7.3 Boot Configuration

External pins BOOTMODE[4:0] select the boot configuration. The values of BOOTMODE[4:0] are latched with the rising edge of RESET. These pins must be valid with the proper setup and hold to this signal. Refer to the data sheet for specific timing requirements.

Table 7–1. Boot Configuration Summary

BOOTMODE [4:0]	Memory Map	Memory at Address 0	Boot Process
00000	MAP 0	SDRAM:4 8-bit devices (SDWID = 0)	None
00001	MAP 0	SDRAM:2 16-bit devices (SDWID = 1)	None
00010	MAP 0	32-bit asynchronous with default timing	None
00011	MAP 0	1/2 rate SBSRAM	None
00100	MAP 0	1x rate SBSRAM	None
00101	MAP 1	Internal	None
00110	MAP 0	External; default values	HPI
00111	MAP 1	Internal	HPI
01000	MAP 0	SDRAM:4 banks of 8-bit memory (SDWID = 0)	8-bit ROM; with default timings
01001	MAP 0	SDRAM:2 banks of 16-bit memory (SDWID = 1)	8-bit ROM; with default timings
01010	MAP 0	32-bit asynchronous with default timing	8-bit ROM; with default timings
01011	MAP 0	1/2 rate SBSRAM	8-bit ROM; with default timings
01100	MAP 0	1x rate SBSRAM	8-bit ROM; with default timings
01101	MAP 1	Internal	8-bit ROM; with default timings
01110		Reserved	
01111		Reserved	

Table 7–1. Boot Configuration Summary (Continued)

BOOTMODE [4:0]	Memory Map	Memory at Address 0	Boot Process
10000	MAP 0	SDRAM 4 banks of 8-bit (SDWID=0)	16-bit ROM; with default timings
10001	MAP 0	SDRAM:2 banks of 16-bit memory (SDWID = 1)	16-bit ROM with default timings
10010	MAP 0	32-bit asynchronous with default timing	16-bit ROM with default timings
10011	MAP 0	1/2 rate SBSRAM	16-bit ROM with default timings
10100	MAP 0	1x rate SBSRAM	16-bit ROM with default timings
10101	MAP 1	Internal	16-bit ROM with default timings
10110		Reserved	
10111		Reserved	
11000	MAP 0	SDRAM:4 banks of 8-bit memory (SDWID = 0)	32-bit ROM with default timings
11001	MAP 0	SDRAM:2 banks of 16-bit memory (SDWID = 1)	32-bit ROM with default timings
11010	MAP 0	32-bit asynchronous with default timing	32-bit ROM with default timings
11011	MAP 0	1/2 rate SBSRAM	32-bit ROM with default timings
11100	MAP 0	1x rate SBSRAM	32-bit ROM with default timings
11101	MAP 1	Internal	32-bit ROM with default timings
11110		Reserved	
11111		Reserved	

7.3.1 Memory Map

The two memory maps, MAP 0 and MAP 1 are summarized in Table 7–2. They differ in that MAP 0 has external memory mapped at address 0, and MAP 1 has internal memory mapped at address 0. Spaces allocated for internal peripherals represent space reserved for that purpose. The entire memory space reserved may not be populated. Refer to the appropriate chapter for the peripherals and internal memory.

Table 7–2. Memory Map Summary

Address Range (Hex)		Size (bytes)	Description of memory block in ...	
			MAP 0	MAP 1
00000000	003FFFFF	4M	External memory interface CE 0	Internal program RAM
00400000	00FFFFFF	12M		External memory interface CE 0
01000000	013FFFFF	4M	External memory interface CE 1	
01400000	017FFFFF	4M	Internal program RAM	External memory interface CE 1
01800000	0183FFFF	256K	Internal peripheral bus EMIF registers	
01840000	0187FFFF	256K	Internal peripheral bus DMA controller registers	
01880000	018BFFFF	256K	Internal peripheral bus HPI register	
018C0000	018FFFFF	256K	Internal peripheral bus MCSP 0 registers	
01900000	0193FFFF	256K	Internal peripheral bus MCSP 1 registers	
01940000	0197FFFF	256K	Internal peripheral bus Timer 0 registers	
01980000	019BFFFF	256K	Internal peripheral bus Timer 1 registers	
019C0000	019FFFFF	256K	Internal peripheral bus Interrupt selector registers	
01A00000	01FFFFFF	6M	Internal peripheral bus Reserved	
02000000	02FFFFFF	16M	External memory interface CE 2	
03000000	03FFFFFF	16M	External memory interface CE 3	
04000000	7FFFFFFF	2G-64M	Reserved	
80000000	803FFFFF	4M	Internal data RAM	
80400000	FFFFFFFF	2G-4M	Reserved	

7.3.2 Memory at Address Reset Address

The boot configuration determines the type of memory located at the reset address for processor operation, address 0, as shown in Table 7–1. When the BOOTMODE [4:0] pins select MAP 1, this memory is internal. When the device mode is in MAP 0, the memory is external. When external memory is selected, BOOTMODE [4:0] also determine the type of memory at the reset address. These options effectively provide alternative reset values to the appropriate EMIF control registers.

7.3.3 Boot Processes

The boot process is determined by the BOOTMODE [4:0] pins as shown in Table 7–1. Three types of boot processes are available:

- No boot process: The CPU begins direct execution from the memory located at address 0. If SDRAM is used in the system, the CPU is held until SDRAM initialization is complete.

- ROM boot process: The memory located at the beginning of CE1 is copied to address 0 by the DMA controller. Although the boot process begins when the device is released from external reset, this transfer occurs while the CPU is internally held in reset. The amount of memory copied is 16K 32-bit words. This boot process also lets you choose the width of the ROM. In this case, the EMIF can automatically assemble consecutive 8-bit bytes or 16-bit halfwords to form the 32-bit instruction words to be moved. These values are expected to be stored in little endian format in the external memory, typically a ROM device.

  The transfer is automatically done by channel 0. The DMA channel is set up to perform an unsynchronized, single-frame block transfer of 16K 32-bit words from the beginning of CE1 to address 0. For SDRAM, the EMIF postpones transfers until the SDRAM has been initialized. The location of CE1 depends on the memory map selected.

  When DMA_INT0 becomes active (exhibits a low-to-high transition), after completion of the block transfer, the CPU is removed from reset and allowed to run from address 0. The DMA_INT0 condition is not latched by the CPU because it occurs while the CPU is still in reset. The DMA_INT0 wakes the CPU from internal reset only if the ROM boot process is selected.

- HPI boot process: The CPU is held in reset while the remainder of the device awakes from reset. During this period, an external host can initialize the CPU's memory space as necessary through the HPI, including external memory configuration registers. Once external memory has been configured, the host can then access any external sections it needs to complete initialization. Once the host is finished with all necessary initialization, it writes a 1 to the DSPINT bit in the HPI control register (HPIC). This write causes an active transition on the DSPINT signal. This transition causes the boot configuration logic to remove the CPU from its reset state. The CPU then begins execution from address 0. The DSPINT condition is not latched by the CPU because it occurs while the CPU is still in reset. Also, DSPINT only wakes the CPU from internal reset only if the HPI boot process is selected.

Chapter 8

Multichannel Buffered Serial Port

Describes the features and operation of the two multichannel buffered serial ports.

Topic **Page**

List of Acronyms

AC97	Audio Codec ‘97 (component specification)
CLKG	Output clock of Sample Rate Generator
CLKGDV	Divider for Sample Rate Generator Clock
CLKR(P/M)	Clock for Receive (Polarity/Mode)
CLKSTP	Clock Stop
CLKX(P/M)	Clock for Transmit (Polarity/Mode)
DLB	Digital Loop Back
DR	Data Receive
DRR	Data Receive Register
DX	Data Transmit
DXR	Data Transmit Register
FPER	Frame Period
FSG	Frame Sync output from sample rate generator
FSGM	FSG Mode
FSR(P/M)	Frame Synchronization (Polarity/Mode) for Receive
FSX(P/M)	Frame Synchronization (Polarity/Mode) for Transmit
FWID	Frame Width
GSYNC	Sample Rate Generator Clock Synchronization
IIS	Inter–IC Sound bus (serial link for digital audio)
IOM	ISDN–Oriented Modular (Architecture and Interfaces)
IOM2	Extended IOM
MCR	Multi–channel Control Register
McBSP	Multi Channel Serial Port
MVIP	Multi–Vendor Integration Protocol
PCR	Pin Control Register
RBR	Receive Buffer Register
RCBLK	Receive Current Block

List of Acronyms *(Continued)*

RCER	Receive Channel Enable Register
RCR	Receive Control Register
RDATDLY	Receive Data Delay
REVT	Receive Synchronization Event to DMA
RFIG	Receive Frame Ignore
RFRLEN(1/2)	Receive Frame Length for Frame Phase 1 or 2
RFULL	RSR (Receive Shift Register) Full
RINT	Receive Interrupt to CPU
RIOEN	Receive I/O Enable
RJUST	Receive data (in RSR) Justification
RMCM	Receive Multi–Channel Mode
RP(A/B)BLK	Receive Partition A/B Block
RPHASE	Receive (number of) Phases
RRDY	Receiver Ready
RRST	Receiver Reset
RSYNCERR	Receive Synchronization Error
RWDLEN(1/2)	Receive Word Length for Frame Phase 1 or 2
SPCR	Serial Port Control Register
SPI	Serial Peripheral Interface (Synchronous)
SRGR	Sample Rate Generator Register
ST–bus	Serial Telecom Bus (Mitel Semiconductor)
XCBLK	Transmit Current Block
XCER	Transmit Channel Enable Register
XCR	Transmit Control Register
XDATDLY	Transmit Data Delay
XEMPTY	XSR (Transmit Shift Register) Empty
XEVT	Transmit Synchronization Event to DMA

List of Acronyms *(Continued)*

XFIG	Transmit Frame Ignore
XFRLEN(1/2)	Transmit Frame Length for Frame Phase 1 or 2
XINT	Transmit Interrupt to CPU
XIOEN	Transmit I/O Enable
XMCM	Transmit Multi–Channel Mode
XP(A/B)BLK	Transmit Partition A/B Block
XPHASE	Transmit (number of) Phases
XRDY	Transmitter Ready
XRST	Transmitter Reset
XSR	Transmit Shift Register

8.1 Features

The Multichannel Serial Port (McBSP) is based on the standard serial port interface found on the TMS320C2x, ’C2xx, ’C5x, and ’C54x devices. Like its predecessors the McBSP provides:

- ❑ Full-Duplex communication
- ❑ Double buffered data registers which allow a continuous data stream
- ❑ Independent framing and clocking for receive and transmit
- ❑ Direct interface to industry standard Codecs, Analog Interface Chips (AICs), and other serially connected A/D and D/A devices
- ❑ External shift clock generation or an internal programmable frequency shift clock
- ❑ Autobuffering capability through the five channel DMA controller.

In addition, the McBSP has the following capabilities:

- ❑ Direct interface to:
  - ■ T1/E1 framers
  - ■ MVIP switching compatible and ST-BUS compliant devices
  - ■ IOM-2 compliant devices
  - ■ AC97 compliant devices. The necessary multi-phase frame synchronization capability is provided
  - ■ IIS compliant devices
  - ■ SPI™ devices
- ❑ Multichannel transmit and receive of up to 128 channels.
- ❑ A wide selection of data sizes including 8-, 12-, 16-, 20-, 24-, or 32-bits

Note:

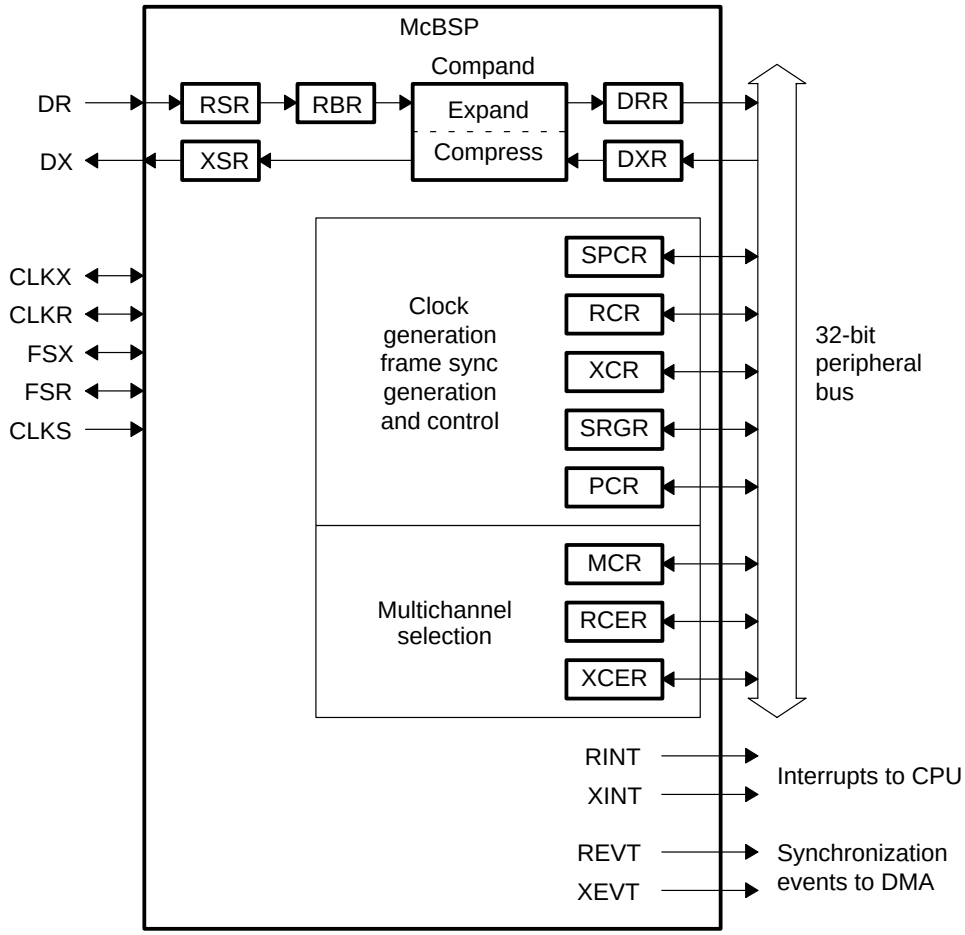
Data sizes are referred to as ‘word’ or ‘serial word’ throughout this document. Therefore when ‘word’ is used, it can be either 8-, 12-, 16-, 20-, 24-, or 32-bits in contrast to the true definition of word as being 32-bits.

- ❑ µ-Law and A-Law companding
- ❑ 8-bit data transfers with LSB or MSB first
- ❑ Programmable polarity for both frame synchronization and data clocks
- ❑ Highly programmable internal clock and frame generation

8.2 General Description

The McBSP consists of a data path and control path as shown in Figure 8–1. Seven pins listed in Table 8–1 connect the control and data paths to external devices.

Figure 8–1. McBSP Internal Block Diagram



Data is communicated to devices interfacing to the McBSP via the Data Transmit (DX) pin for transmission and the Data Receive (DR) pin for reception. Control information in the form of clocking and frame synchronization is communicated via CLKX, CLKR, FSX, and FSR. The ’C6x communicates to the McBSP via 32-bit wide control registers accessible via the internal peripheral bus. Either the CPU or DMA reads the received data from the Data Receive Register (DRR) and writes the data to be transmitted to the Data Transmit Register (DXR). Data written to the DXR is shifted out to DX via the Transmit Shift Register (XSR). Similarly, receive data on the DR pin is shifted into RSR and copied into RBR. RBR is then copied to DRR which can be read by the CPU or DMA. This allows internal data movement and external data communications simultaneously. The remaining registers which are accessible to the CPU configure the control mechanism of the McBSP. These registers are listed in Table 8–2. The control block consists of internal clock generation, frame synchronization signal generation, and their control, and multichannel selection. This control block sends notification of important events to the CPU and DMA via four signals shown in Table 8–3.

Table 8–1. McBSP Interface Signals

Pin	I/O/Z	Description
CLKR	I/O/Z	Receive clock
CLKX	I/O/Z	Transmit clock
CLKS	I	External clock
DR	I	Received serial data
DX	O/Z	Transmitted serial data
FSR	I/O/Z	Receive frame synchronization
FSX	I/O/Z	Transmit frame synchronization

Table 8–2. McBSP Registers

Hex Byte Address				
McBSP 0	**McBSP 1**	**Acronym**	**McBSP Register Name†**	**Section**
–	–	RBR	Receive Buffer Register	8.2
–	–	RSR	Receive Shift Register	8.2
–	–	XSR	Transmit Shift Register	8.2
018C0000	01900000	DRR	Data Receive Register‡	8.2
018C0004	01900004	DXR	Data Transmit Register	8.2
018C0008	01900008	SPCR	Serial Port Control Register	8.2.1
018C000C	0190000C	RCR	Receive Control Register	8.2.2
018C0010	01900010	XCR	Transmit Control Register	8.2.2
018C0014	01900014	SRGR	Sample Rate Generator Register	8.5.1.1
018C0018	01900018	MCR	Multichannel Register	8.6.1
018C001C	0190001C	RCER	Receive Channel Enable Register	8.6.3.1
018C0020	01900020	XCER	Transmit Channel Enable Register	8.6.3.1
018C0024	01900024	PCR	Pin Control Register	8.2.1 and 8.8

† The RBR, RSR, and XSR are not directly accessible via the CPU or DMA.
‡ This register is read-only to the CPU and DMA.

Table 8–3. McBSP CPU Interrupts and DMA Event Synchronization

Interrupt Name	Description	Section
RINT	Receive Interrupt to CPU	8.3.3
XINT	Transmit Interrupt to CPU	8.3.3
REVT	Receive Synchronization Event to DMA	8.3.2.1
XEVT	Transmit Synchronization Event to DMA	8.3.2.2

8.2.1 Serial Port Configuration

The serial port is configured via the 32-bit Serial Port Control Register (SPCR) and Pin Control Register (PCR) as shown in Figure 8–2 and Figure 8–3. The SPCR and PCR contain McBSP status information and also bits that can be configured for desired operation. The operation of each bit-field will be discussed in the section listed in the bit-field description in Table 8–4 and Table 8–5.

Figure 8–2. Serial Port Control Register (SPCR)

31–24	23	22	21–20	19	18	17	16
reserved†	$\overline{\text{FRST}}$	$\overline{\text{GRST}}$	XINTM	XSYNCERR‡	$\overline{\text{XEMPTY}}$	XRDY	$\overline{\text{XRST}}$
R, +0	RW, +0	RW, +0	RW, +0	RW, +0	R, +0	R, +0	RW, +0

15	14–13	12–10	9–8	7	6	5–4	3	2	1	0
DLB	RJUST	CLKSTP	reserved	reserved	reserved	RINTM	RSYNCERR	RFULL	RRDY	$\overline{\text{RRST}}$
RW,+0	RW, +0	RW,+0	R, +0	R, +0	R, +0	RW, +0	RW, +0	R, +0	R, +0	RW, +0§

† This and all reserved bit-fields have *no* storage associated with them. However, they are always read as 0.
‡ **Caution**: Writing a '1' to this bit will set the error condition. Thus, it is mainly used for testing purposes or if this operation is desired.
§ R +0 means read only, reset value is zero. RW, +0 indicates read and write allowed, reset value is zero.

Table 8–4. Serial Port Control Register (SPCR) Bit-Field Description

Name	Function	Section
$\overline{\text{FRST}}$	Frame Sync Generator Reset $\overline{\text{FRST}}$ = 0, The frame sync generation logic is reset. Frame sync signal is not generated by the sample rate generator. $\overline{\text{FRST}}$ = 1, Frame sync signal is generated after (FPER + 1) number of CLKG clocks. All frame counters are loaded (FPER, FWID) with their programmed values.	
$\overline{\text{GRST}}$	Sample Rate Generator Reset $\overline{\text{GRST}}$ = 0, Sample rate generator is reset. $\overline{\text{GRST}}$ = 1, Sample rate generator is pulled out of reset; CLKG is driven as per programmed values in Sample Rate Generator Register (SRGR).	8.5.1.2
RINTM XINTM	Receive/Transmit Interrupt mode (R/X)INTM = 00b, (R/X)INT driven by (R/X)RDY (R/X)INTM = 01b, (R/X)INT generated by end-of-block or end-of-frame in multi-channel operation (R/X)INTM = 10b, (R/X)INT generated by a new frame synchronization (R/X)INTM = 11b, (R/X)INT generated by (R/X)SYNCERR	8.3.3
RSYNCERR XSYNCERR	Receive/Transmit Synchronization Error (R/X)SYNCERR = 0, no frame synchronization error (R/X)SYNCERR = 1, frame synchronization error detected by McBSP.	8.3.7.2 8.3.7.5
$\overline{\text{XEMPTY}}$	Transmit Shift Register (XSR) Empty $\overline{\text{XEMPTY}}$ = 0, XSR is empty $\overline{\text{XEMPTY}}$ = 1, XSR is not empty	8.3.7.4
RFULL	Receive Shift Register (RSR) Full error condition RFULL = 0, Receiver is not in overrun condition RFULL = 1, DRR is not read, RBR is full, and RSR is also full with new word.	8.3.7.1
RRDY XRDY	Receiver/Transmitter Ready (R/X)RDY = 0, receiver/transmitter is not ready. (R/X)RDY = 1, receiver is ready with data to be read from DRR or transmitter is ready with data in DXR.	8.3.2
$\overline{\text{RRST}}$ $\overline{\text{XRST}}$	Receiver/transmitter reset. This resets and enables the receiver/transmitter. $\overline{\text{(R/X)RST}}$ = 0, The serial port receiver/transmitter is disabled and in reset state. $\overline{\text{(R/X)RST}}$ = 1, The serial port receiver/transmitter is enabled.	8.3.1

Table 8–4. Serial Port Control Register (SPCR) Bit-Field Description (Continued)

Name	Function	Section
DLB	Digital Loop Back Mode DLB = 0, Digital loop back mode disabled DLB = 1, Digital loop back mode enabled	8.5.2.5 8.5.2.6 8.5.3.2
RJUST	Receive Sign-Extension and Justification Mode RJUST = 00b, right-justify and zero-fill MSBs in DRR RJUST = 01b, right-justify and sign-extend MSBs in DRR RJUST = 10b, left-justify and zero-fill LSBs in DRR RJUST = 11b, reserved	8.3.8
CLKSTP	Clock Stop Mode CLKSTP = 0Xb, Clock Stop Mode Disabled. Normal clocking enabled for non-SPI mode. Various SPI modes (in conjunction with CLKXP in PCR, CLKRP = X) when: CLKSTP = 10b and CLKXP = 0, Clock starts with rising edge without delay CLKSTP = 10b and CLKXP = 1, Clock starts with falling edge without delay CLKSTP = 11b and CLKXP = 0, Clock starts with rising edge with delay CLKSTP = 11b and CLKXP = 1, Clock starts with falling edge with delay	8.7

In addition to PCR being used to configure the McBSP pins as inputs or outputs during normal serial port operation, it is used to configure the serial port pins as general purpose inputs or outputs during receiver and/or transmitter reset. This is described in Section 8.8.

Figure 8–3. Pin Control Register (PCR)

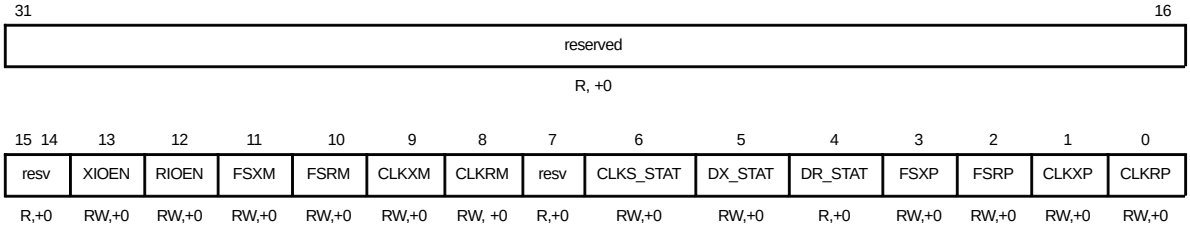


Table 8–5. Pin Control Register (PCR) Bit-Field Description

Name	Function	Section
XIOEN RIOEN	Transmit/Receive General Purpose I/O Mode ONLY when (R/X)$\overline{\text{RST}}$ = 0 in SPCR (R/X)IOEN = 0, DR and CLKS pins are not general purpose inputs; DX pin is not a general purpose output; FS(R/X), CLK(R/X) are not general purpose I/Os. (R/X)IOEN = 1, DR and CLKS pins are general purpose inputs; DX pin is a general purpose output. FS(R/X), CLK(R/X) are general purpose I/Os. These serial port pins do not perform serial port operation. The CLKS pin is affected by a combination of $\overline{\text{RST}}$ and IOEN signals of the receiver and transmitter.	8.8
FSXM	Transmit Frame Synchronization Mode FSXM = 0, Frame synchronization signal derived from an external source FSXM = 1, Frame synchronization is determined by the Sample Rate Generator frame synchronization mode bit FSGM in the SRGR.	8.5.3.3 and 8.8
FSRM	Receive Frame Synchronization Mode FSRM = 0, Frame synchronization pulses generated by an external device. FSR is an input pin FSRM = 1, Frame synchronization generated internally by sample rate generator. FSR is an output pin except when GSYNC = 1 (subsection 8.5.1.1) in SRGR.	8.5.3.2 and 8.8
CLKRM	Receiver Clock Mode Case 1: Digital Loop Back Mode not set (DLB = 0) in SPCR CLKRM = 0, Receive clock (CLKR) is an input driven by an external clock. CLKRM = 1, CLKR is an output pin and is driven by the internal sample rate generator. Case 2: Digital Loop Back Mode set (DLB = 1) in SPCR CLKRM = 0, Receive clock (not the CLKR pin) is driven by transmit clock (CLKX) which is based on CLKXM bit in PCR. CLKR pin is in high impedance. CLKRM = 1, CLKR is an output pin and is driven by the transmit clock. The transmit clock is derived based on CLKXM bit in the PCR.	8.5.2.6 and 8.8

Table 8–5. Pin Control Register (PCR) Bit-Field Description (Continued)

Name	Function	Section
CLKXM	Transmitter Clock Mode. CLKXM = 0, Receiver/Transmitter clock is driven by an external clock with CLK(R/X) as an input pin. CLKXM = 1, CLK(R/X) is an output pin and is driven by the internal sample rate generator. During SPI mode (CLKSTP in SPCR is a non-zero value): CLKXM = 0, McBSP is a slave and (CLKX) is driven by the SPI master in the system. CLKR is internally driven by CLKX. CLKXM = 1, McBSP is a master and generates the transmitter clock (CLKX) to drive its receiver clock (CLKR) and the shift clock of the SPI-compliant slaves in the system.	8.5.2.7 and 8.8 8.7
CLKS_ STAT	CLKS pin status. Reflects value on CLKS pin when selected as a general purpose input.	8.8
DX_STAT	DX pin status. Reflects value driven on to DX pin when selected as a general purpose output.	8.8
DR_STAT	DR pin status. Reflects value on DR pin when selected as a general purpose input.	8.8
FSXP FSRP	Receive/Transmit Frame Synchronization Polarity FS(R/X)P = 0, Frame synchronization pulse FS(R/X) is active high FS(R/X)P = 1, Frame synchronization pulse FS(R/X) is active low	8.3.4.1 and 8.8
CLKXP	Transmit Clock Polarity CLKXP = 0, Transmit data driven on rising edge of CLKX CLKXP = 1, Transmit data driven on falling edge of CLKX	8.3.4.1 and 8.8
CLKRP	Receive Clock Polarity CLKRP = 0, Receive data sampled on falling edge of CLKR CLKRP = 1, Receive data sampled on rising edge of CLKR	8.3.4.1 and 8.8

8.2.2 Receive and Transmit Control Registers: RCR and XCR

The Receive and Transmit Control Registers (RCR and XCR) shown in Figure 8–4 and Figure 8–5, configure various parameters of receive and transmit operation respectively. The operation of each bit-field will be discussed in the section listed in the bit-field description in Table 8–6.

Figure 8–4. Receive Control Register (RCR)

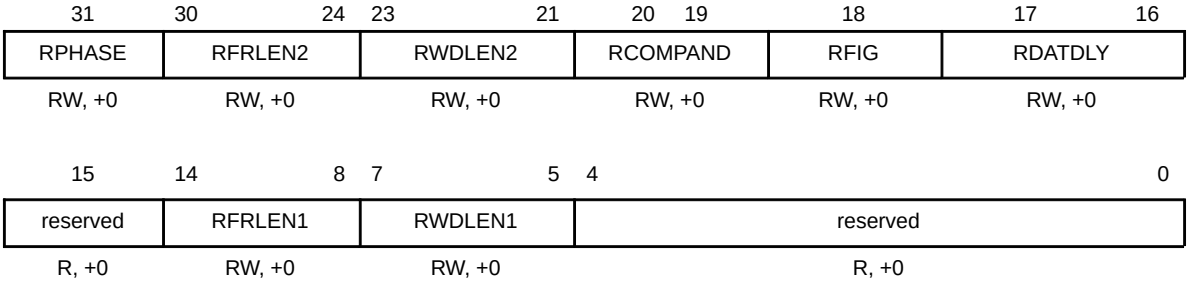


Figure 8–5. Transmit Control Register (XCR)

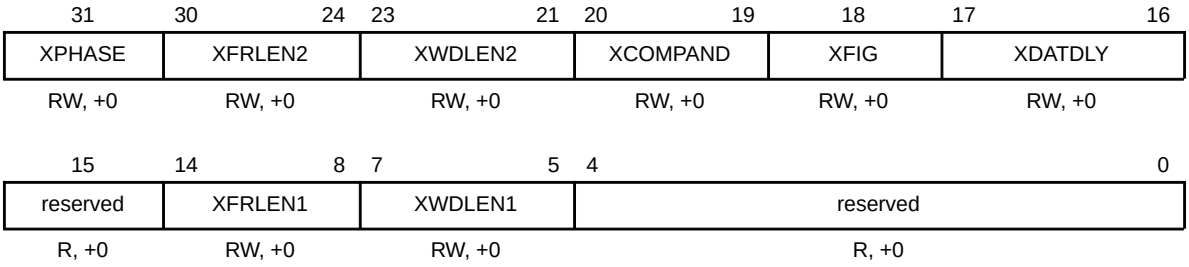


Table 8–6. Receive/Transmit Control Register (RCR/XCR) Bit-Field Description

Name	Function	Section
(R/X)PHASE	Receive/Transmit Phases (R/X)PHASE = 0, single phase frame (R/X)PHASE = 1, dual phase frame	8.3.4.2
(R/X)FRLEN(1/2)	Receive/Transmit Frame Length 1/2 (R/X)FRLEN(1/2) = 000 0000b, 1 word per frame (R/X)FRLEN(1/2) = 000 0001b, 2 words per frame | | (R/X)FRLEN(1/2) = 111 1111b, 128 words per frame	8.3.4.3
(R/X)WDLEN(1/2)	Receive/Transmit Word Length 1/2 (R/X)WDLEN(1/2) = 000b, 8 bits (R/X)WDLEN(1/2) = 001b, 12 bits (R/X)WDLEN(1/2) = 010b, 16 bits (R/X)WDLEN(1/2) = 011b, 20 bits (R/X)WDLEN(1/2) = 100b, 24 bits (R/X)WDLEN(1/2) = 101b, 32 bits (R/X)WDLEN(1/2) = 11Xb, reserved	8.3.4.4
RCOMPAND XCOMPAND	Receive/Transmit Companding Mode. Modes other than 00b are only enabled when the appropriate (R/X)WDLEN is 000b, indicating 8-bit data. (R/X)COMPAND = 00b, no companding, data transfer starts with MSB first. (R/X)COMPAND = 01b, no companding, 8-bit data, transfer starts with LSB first. (R/X)COMPAND = 10b, compand using μ-law for receive/transmit data. (R/X)COMPAND = 11b, compand using A-law for receive/transmit data.	NO TAG

Table 8–6. Receive/Transmit Control Register (RCR/XCR) Bit-Field Description (Continued)

Name	Function	Section
RFIG XFIG	Receive/Transmit Frame Ignore (R/X)FIG = 0, Receive/Transmit Frame synchronization pulses after the first restarts the transfer. (R/X)FIG = 1, Receive/Transmit Frame synchronization pulses after the first are ignored.	8.3.6.2
RDATDLY XDATDLY	Receive/Transmit data delay (R/X)DATDLY = 00b, 0-bit data delay (R/X)DATDLY = 01b, 1-bit data delay (R/X)DATDLY = 10b, 2-bit data delay (R/X)DATDLY = 11b, reserved	8.3.4.6

8.3 Data Transmission and Reception Flow

As shown in Figure 8–1, the receive operation is triple-buffered and transmit operation is double buffered. Receive data arrives on DR and is shifted into the RSR. Once a full word (8-, 12-, 16-, 20-, 24-, or 32-bit) is received, the RSR is copied to the Receive Buffer Register, RBR, only if RBR is not full. RBR is then copied to DRR unless DRR is not read by the CPU or DMA.

Transmit data is written by the CPU or DMA to the DXR. If there is no data in the XSR, the value in the DXR is copied to the XSR. Otherwise, the DXR is copied to the XSR when the last bit of data is shifted out on DX. After transmit frame synchronization, the XSR begins shifting out the transmit data on DX.

8.3.1 Resetting the Serial Port: $\overline{\text{(R/X)RST}}$, $\overline{\text{GRST}}$, and $\overline{\text{RESET}}$

The serial port can be reset in the following two ways:

1) Device reset ($\overline{\text{RESET}}$ = 0) places the receiver, transmitter and the sample rate generator in reset. When the device reset is removed ($\overline{\text{RESET}}$ = 1), $\overline{\text{FRST}}$ = $\overline{\text{GRST}}$ = $\overline{\text{RRST}}$ = $\overline{\text{XRST}}$ = 0, keeping the entire serial port in the reset state.

2) The serial port transmitter and receiver can be independently reset by the $\overline{\text{XRST}}$ and $\overline{\text{RRST}}$ bits in the Serial Port Control register. The sample rate generator is reset by the $\overline{\text{GRST}}$ bit in the SPCR.

Table 8–7 shows the state of McBSP pins when the serial port is reset due to device reset and due to receiver/transmitter reset ($\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = $\overline{\text{FRST}}$ = 0).

Table 8–7. Reset State of McBSP Pins

McBSP Pins	Direction	Device Reset ($\overline{\text{RESET}}$ = 0)	McBSP Reset
			Receiver Reset ($\overline{\text{RRST}}$ = 0 and $\overline{\text{FRST}}$ = 0)
DR	I	Known State†	Known State†
CLKR	I/O/Z	Known State†	Known State if Input†; CLKRP if Output†
FSR	I/O/Z	Known State†	Known State if Input†; FSRP(inactive state) if Output†
CLKS	I	Known State†	Known State if the transmitter is also reset or if the transmitter is configured with CLKSM = 1 in SRGR. See subsection 8.5.1.1†
			Transmitter Reset ($\overline{\text{XRST}}$ = 0 and $\overline{\text{FRST}}$ = 0)
DX	O	Hi-Z	Hi-Z
CLKX	I/O/Z	Known State†	Known State if Input; CLKX if Output†
FSX	I/O/Z	Known State†	Known State if Input; FSXP(inactive state) if Output†
CLKS	I	Known State†	Known State if the receiver is also reset or if the receiver is configured with CLKSM = 1 in SRGR. See subsection 8.5.1.1†

† All input pins should be pulled up with individual resistors.

- **Device reset or McBSP reset:** When the McBSP is reset in any of the above two ways, the state machine is reset to its initial state. This initial state includes resetting all counters and status bits. The receive status bits include RFULL, RRDY, and RSYNCERR. The transmit status bits include $\overline{\text{XEMPTY}}$, XRDY, and XSYNCERR.

- **Device reset:** When McBSP is reset due to device reset (device pin $\overline{\text{RESET}}$ = 0), the entire serial port including the transmitter, receiver, and the sample rate generator is reset. All input only pins and three-state pins should be in a known state†. The output only pin, DX, is in high impedance state. Since the sample rate generator is also reset ($\overline{\text{GRST}}$ = 0), the sample rate generator clock, CLKG, is driven by a divide-by-2 CPU clock, whereas the frame sync signal, FSG, is not generated. See subsection 8.5.1.2 for more information on sample rate generator reset. When the device is pulled out of reset, the serial port remains in reset condition ((R/X)$\overline{\text{RST}}$ = $\overline{\text{FRST}}$ = 0), and in this condition the DR and DX pins may be used as general purpose I/O as described in subsection 8.8.

- **McBSP reset:** When the receiver and transmitter reset bits, $\overline{\text{RRST}}$ and $\overline{\text{XRST}}$, are written with a zero value, the respective portions of the McBSP are reset. Activity in the corresponding section of the serial port stops. The $\overline{\text{FRST}}$ bit in SPCR is set to zero which resets the frame sync generation counters in the sample generator. All input only pins such as DR, and CLKS, and all other pins that are configured as inputs are in a known state. FS(R/X) is driven to its inactive state (same as its polarity bit FS(R/X)P) if they are outputs. CLKR and CLKX are driven by the sample rate generator. If CLK(R/X) are programmed as outputs, they are driven by CLKG. Lastly, the DX pin will be in high impedance state when the transmitter is reset. During normal operation (out of device reset), the sample rate generator can be reset by writing a zero to $\overline{\text{GRST}}$. $\overline{\text{GRST}}$ should be low only when neither the transmitter nor the receiver is using the sample rate generator. In this case, the internal sample rate generator clock CLKG, and its frame sync signal (FSG) is driven inactive low. When the sample rate generator is not in reset state ($\overline{\text{GRST}}$ = 1), the pins FSR and FSX will be in an inactive state when $\overline{\text{RRST}}$ = 0 and $\overline{\text{XRST}}$ = 0 respectively, even if they are outputs driven by FSG. This ensures that when only one portion of the McBSP is in reset, the other portion can continue operation when $\overline{\text{FRST}}$ = 1 and frame sync is driven by FSG. See subsection 8.5.1.2 for more information on sample rate generator reset.

- **Sample Rate Generator Reset:** As mentioned earlier, the sample rate generator is reset when the device is reset or when its reset bit, $\overline{\text{GRST}}$, is written with a value of zero. In the case of device reset, the CLKG signal is driven by a divide-by-2 CPU clock and FSG is driven inactive low. If it is desired to reset the sample rate generator when neither the transmitter or receiver is fed by the CLKG and FSG, $\overline{\text{GRST}}$ in the SRGR can be programmed to zero. Here, CLKG and FSG are driven inactive low. When $\overline{\text{GRST}}$ = 1, CLKG comes up running as programmed in the SRGR. Later if $\overline{\text{FRST}}$ = 1, FSG is driven active high after the programmed frame period (FPER+1) number of CLKG cycles have elapsed.

After device reset is complete ($\overline{\text{RESET}}$ = 1), the serial port initialization procedure is as follows:

1) Set $\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = $\overline{\text{FRST}}$ = 0 in SPCR. If coming out of device reset, this step is not required.

2) Program only the McBSP configuration registers (and not the data registers) listed in Table 8–2 as required when the serial port is in reset state ($\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = $\overline{\text{FRST}}$ = 0).

3) Wait two bit clocks. This is to ensure proper synchronization internally.

4) Set $\overline{XRST}$ = $\overline{RRST}$ = 1 to enable the serial port. Note that the value written to the SPCR at this time should have only the reset bits changed to 1 and the remaining bit-fields should have the same value as in Step 2 above.

5) Set up data acquisition as desired.

6) Set $\overline{FRST}$ = 1. Now the McBSP is ready to transmit and/or receive, if it is the frame master.

Alternatively, on either write (Steps 1 and 4 above), the transmitter and receiver may be placed in or taken out of reset individually by only modifying the desired bit. Note that the necessary duration of the active-low period of $\overline{XRST}$ or $\overline{RRST}$ is at least two bit clocks (CLKR/CLKX) wide. The above procedure for reset initialization can be applied in general when the receiver or transmitter has to be reset during its normal operation and also when the sample rate generator is not used for either operation. The sample rate generator reset procedure is explained in subsection 8.5.1.2.

Note:

(a) The appropriate bit-fields in the serial port configuration registers SPCR, PCR, RCR, XCR, and SRGR should only be modified by the user when the affected portion of the serial port is in reset.

(b) Data Transmit Register, DXR, should be loaded by the CPU or DMA only when the transmitter is not in reset ($\overline{XRST}$ = 1). Exception to this rule is during Digital Loop Back Mode which is described in subsection 8.4.1

(c) The multichannel selection registers MCR, XCER, and RCER can be modified at any time as long as they are not being used by the current block in the multichannel selection; see subsection 8.6.3.2 for further details in this case.

8.3.2 Determining Ready Status

RRDY and XRDY indicate the ready state of the McBSP receiver and transmitter, respectively. Writes and reads of the serial port may be synchronized by polling RRDY and XRDY, or by using the events to DMA (REVT and XEVT) or interrupts to CPU (RINT and XINT) that they generate. Note that reading the DRR and writing to DXR affects RRDY and XRDY.

8.3.2.1 Receive Ready Status: REVT, RINT, and RRDY

RRDY = 1 indicates that the RBR contents have been copied to the DRR and that the data can be read by either the CPU or DMA. Once that data has been read by either the CPU or DMA, RRDY is cleared to 0. Also, at device reset

or serial port receiver reset ($\overline{\text{RRST}}$ = 0), the RRDY is cleared to 0 to indicate no data has yet been received and loaded into DRR. RRDY directly drives the McBSP receive event to the DMA (REVT). Also, the McBSP receive interrupt (RINT) to the CPU may be driven by RRDY if RINTM = 00b (default value) in the SPCR.

8.3.2.2 Transmit Ready Status: XEVT, XINT, and XRDY

XRDY = 1 indicates that the DXR contents have been copied to XSR and that DXR is ready to be loaded with a new data word. When the transmitter transitions from reset to non-reset ($\overline{\text{XRST}}$ transitions from 0 to 1), the XRDY also transitions from 0 to 1 indicating that the DXR is ready for new data. Once new data is loaded by the CPU or DMA, XRDY is cleared to 0. However, once this data is copied from the DXR to the XSR, XRDY transitions again from 0 to 1. Now again, the CPU or DMA can write to DXR although XSR has not been shifted out on DX as yet. XRDY directly drives the transmit synchronization event to the DMA (XEVT). Also, the transmit interrupt (XINT) to the CPU may also be driven by XRDY if XINTM = 00b (default value) in the SPCR.

8.3.3 CPU Interrupts: (R/X)INT

The receive interrupt (RINT) and transmit interrupt (XINT) signals the CPU of changes to the serial port status. Four options exist for configuring these interrupts. These options are set by the receive/transmit interrupt mode bit-field, (R/X)INTM, in the SPCR.

1) (R/X)INTM = 00b. Interrupt on every serial word by tracking the (R/X)RDY bits in the SPCR. Sections 8.3.2.1 and 8.3.2.2 describe the RRDY and XRDY bits.

2) (R/X)INTM = 01b. Interrupt after every 16-channel block boundary (in multichannel selection mode) has been crossed within a frame. In any other serial transfer case, this setting is not applicable and therefore no interrupts are generated. See subsection 8.6.3.3 for details.

3) (R/X)INTM = 10b. Interrupt on detection of frame synchronization pulses. This generates an interrupt even when the transmitter/receiver is in reset. This is done by synchronizing the incoming frame sync pulse to the CPU clock and sending it to the CPU via (R/X)INT. This is described in subsection 8.5.3.4.

4) (R/X)INTM = 11b. Interrupt on frame synchronization error. Note that if any of the other interrupt modes are selected, (R/X)SYNCERR may be read to detect this condition. See subsections 8.3.7.2 and 8.3.7.5 for more details on synchronization error.

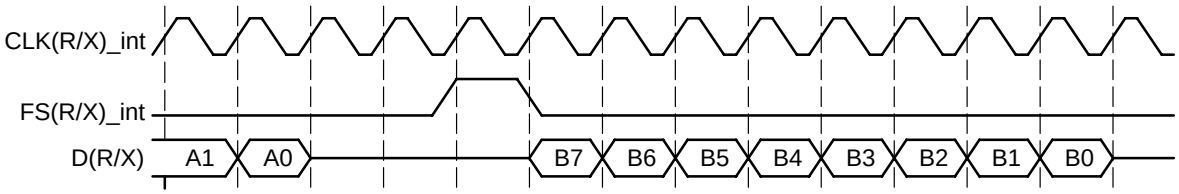
The last three options listed (2–4) are applicable as interrupts to the CPU, and not as events to the DMA.

8.3.4 Frame and Clock Configuration

Figure 8–6 shows typical operation of the McBSP clock and frame sync signals. Serial clocks CLKR, and CLKX define the boundaries between bits for receive and transmit respectively. Similarly, frame sync signals FSR and FSX define the beginning of a serial word. The McBSP allows configuration of various parameters for data frame synchronization. This can be done independently for receive and transmit which includes the following:

- Polarities of FSR, FSX, CLKX, and CLKR may be independently programmed.
- A choice of single or dual-phase frames.
- For each phase, the number of words per frame is programmable.
- For each phase, the number of bits per word is programmable.
- Subsequent frame synchronization may restart the serial data stream or be ignored.
- The data delay from frame synchronization to first data bit can be 0-, 1-, or 2-bit delays.
- Right or left-justification as well as sign-extension or zero-filling can be chosen for receive data.

Figure 8–6. Frame and Clock Operation



8.3.4.1 Frame and Clock Operation

Receive and transmit frame sync pulses can be generated either internally by the sample rate generator (See subsection 8.5.1) or driven by an external source. This can be achieved by programming the mode bit, FS(R/X)M, in the PCR. FSR is also affected by the GSYNC bit in the SRGR (See subsection 8.5.3.2 for details). Similarly, receive and transmit clocks can be selected to be inputs or outputs by programming the mode bit, CLK(R/X)M, in the PCR.

When FSR and FSX are inputs (FSXM = FSRM = 0, external frame sync pulses), the McBSP detects them on the internal falling edge of clock, CLKR_int, and CLKX_int respectively (See Figure 8–35). The receive data arriving at DR pin is also sampled on the falling edge of CLKR_int. Note that these internal clock signals are either derived from external source via CLK(R/X) pins or driven by the sample rate generator clock (CLKG) internal to the McBSP.

When FSR and FSX are outputs driven by the sample rate generator, they are generated (transition to their active state) on the rising edge of internal clock, CLK(R/X)_int. Similarly data on DX pin is output on the rising edge of CLKX_int. See subsection 8.3.4.6 for further details.

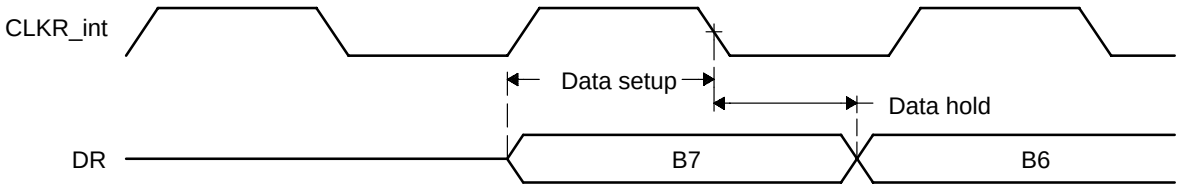
FSRP, FSXP, CLKRP, and CLKXP configure the polarities of FSR, FSX, CLKR, and CLKX signals as shown in Table 8–5. All frame sync signals (FSR_int, FSX_int) internal to the serial port are active high. If the serial port is configured for external frame synchronization (FSR/FSX are inputs to McBSP), and FSRP = FSXP = 1, the external active low frame sync signals are inverted before being sent to the receiver (FSR_int) and transmitter (FSX_int). Similarly, if internal synchronization (FSR/FSX are output pins and GSYNC = 0) is selected, the internal active high sync signals are inverted if the polarity bit FS(R/X)P = 1, before being sent to the FS(R/X) pin. Figure 8–35 shows this inversion using XOR gates.

On the transmit side, the transmit clock polarity bit, CLKXP, sets the edge used to shift and clock out transmit data. Note that data is always transmitted on the rising edge of CLKX_int. If CLKXP = 1, and external clocking is selected (CLKXM = 0 and CLKX is an input), the external falling-edge triggered input clock on CLKX is inverted to a rising-edge triggered clock before being sent to the transmitter. If CLKXP = 1, and internal clocking selected (CLKXM = 1 and CLKX is an output pin), the internal (rising-edge triggered) clock, CLKX_int, is inverted before being sent out on the CLKX pin.

Similarly, on the transmitter side, the configuration is such that the receiver can reliably sample data that is clocked with a rising edge clock (by the transmitter). The receive clock polarity bit, CLKRP, sets the edge used to sample received data. Note that the receive data is always sampled on the falling edge of CLKR_int. Therefore, if CLKRP = 1 and external clocking is selected (CLKRM = 0 and CLKR is an input pin), the external rising edge triggered input clock on CLKR is inverted to a falling-edge before being sent to the receiver. If CLKRP = 1, and internal clocking is selected (CLKRM = 1), the internal falling-edge triggered clock is inverted to a rising-edge before being sent out on the CLKR pin.

Note that in a system where the same clock (internal or external) is used to clock the receiver and transmitter, CLKRP = CLKXP. The receiver uses the opposite edge as the transmitter to guarantee valid setup and hold of data around this edge. Figure 8–7 shows how data clocked by an external serial device using a rising edge may be sampled by the McBSP receiver with the falling edge of the same clock.

Figure 8–7. Receive Data Clocking



8.3.4.2 Frame Synchronization Phases

Frame synchronization is used to indicate the beginning of a transfer on the McBSP. The data stream following frame synchronization may have two phases, phase 1 and phase 2. The number of phases in a data stream can be selected by the phase bit, (R/X)PHASE, in the RCR and XCR. The number of words per frame and bits per word can be independently selected for each phase via (R/X)FRLEN(1/2) and (R/X)WDLEN(1/2) respectively. Figure 8–8 shows an example of a frame where the first phase consists of 2 words of 12 bits each followed by a second phase of 3 words of 8 bits each. Note that the entire bit stream in the frame is contiguous. There are no gaps either between words or phases. Table 8–8 shows the bit-fields in the receive/transmit control register (RCR/XCR) that control the number of words/frame and bits/word for each phase for both the receiver and transmitter. The maximum number of words per frame is 128 for a single phase frame and 256 for a dual phase frame. The number of bits per word can be 8-, 12-, 16-, 20-, 24-, or 32-bits.

Figure 8–8. Dual Phase Frame Example

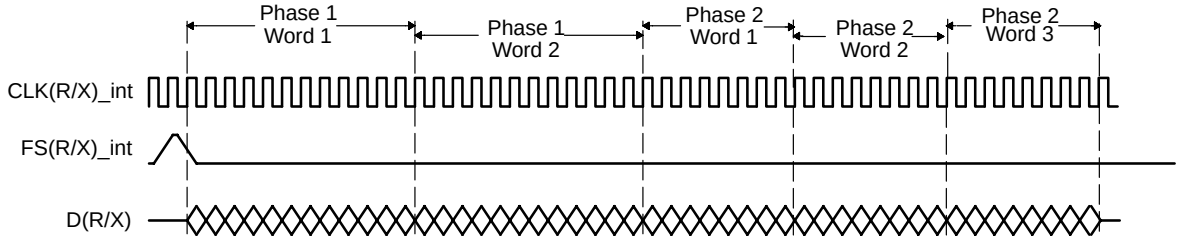


Table 8–8. RCR/XCR Bit-Fields Controlling Words/Frame and Bits/Word

Serial Port McBSP0/1	Frame Phase	RCR/XCR Bit-field Control	
		Words/Frame	Bits/Word
Receive	1	RFRLEN1	RWDLEN1
Receive	2	RFRLEN2	RWDLEN2
Transmit	1	XFRLEN1	XWDLEN1
Transmit	2	XFRLEN2	XWDLEN2

8.3.4.3 Frame Length: (R/X)FRLEN(1/2)

Frame length can be defined as the number of serial words (8-, 12-, 16-, 20-, 24-, or 32-bit) transferred per frame. The length corresponds to the number of words or logical time slots or channels per frame synchronization signal. The 7-bit (R/X)FRLEN(1/2) field in the (R/X)CR supports up to 128 words per frame as shown in Table 8–9. Note that (R/X)PHASE = 0 represents a single phase data frame and a (R/X)PHASE = 1 selects a dual phase for the data stream. Note that for a single phase frame, FRLEN2 is a don't care. The user is cautioned to program the frame length fields with (*w minus 1)*, where *w* represents the number of words per frame. For the example in Figure 8–8, (R/X)FRLEN1 = 1 or 0000001b and (R/X)FRLEN2 = 2 or 0000010b.

Table 8–9. McBSP Receive/Transmit Frame Length 1/2 Configuration

(R/X)PHASE	(R/X)FRLEN1	(R/X)FRLEN2	Frame Length
0	$0 \le n \le 127$	X	Single Phase Frame; (n+1) words per frame
1	$0 \le n \le 127$	$0 \le m \le 127$	Dual Phase Frame; (n+1) plus (m+1) words per frame

8.3.4.4 Word Length: (R/X)WDLEN(1/2)

The (R/X)WDLEN(1/2) fields in the receive/transmit control register determine the word length in bits per word for the receiver and transmitter for each phase of the frame as shown in Table 8–8. Table 8–10 shows how the value of these fields selects particular word lengths in bits. For the example in Figure 8–8, (R/X)WDLEN1 = 001b, and (R/X)WDLEN2 = 000b. Note that if (R/X)PHASE = 0 indicating a single phase frame, R/X)WDLEN2 is not used by the McBSP and its value is a don't care.

Table 8–10. McBSP Receive/Transmit Word Length Configuration

(R/X)WDLEN(1/2)	McBSP Word Length (Bits)
000	8
001	12
010	16
011	20
100	24
101	32
110	reserved
111	reserved

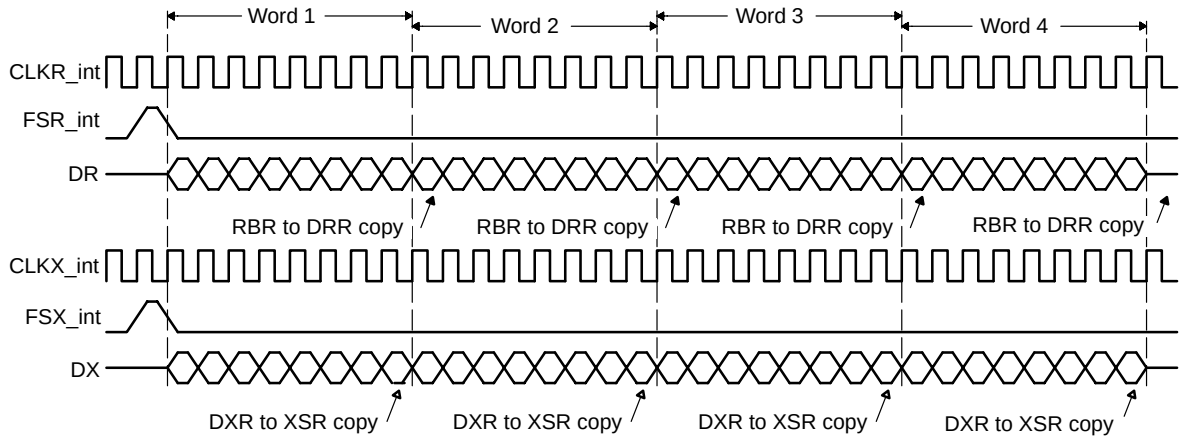
8.3.4.5 Data Packing using Frame Length and Word Length

The frame length and word length can be manipulated to effectively pack data. For example, consider a situation where four 8-bit words are transferred in a single phase frame as shown in Figure 8–9. In this case:

- (R/X)FRLEN1 = 0000011b, 4-word frame
- (R/X)PHASE = 0, single phase frame
- (R/X)FRLEN2 = X
- (R/X)WDLEN1 = 000b, 8-bit words

In this case, four 8-bit data elements are transferred to and from the McBSP by the CPU or DMA. Thus, four reads of DRR and four writes of DXR are necessary for each frame.

Figure 8–9. Single Phase Frame of Four 8-Bit Words

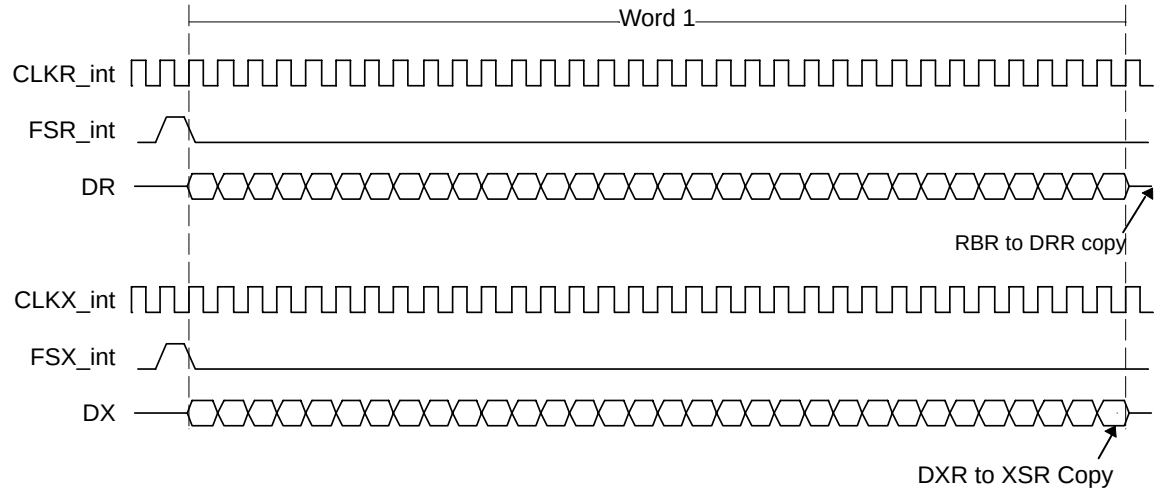


The example in Figure 8–9 can also be viewed as a data stream of a single phase frame consisting of one 32-bit data word as shown in Figure 8–10. In this case:

- (R/X)FRLEN1 = 0b, 1-word frame
- (R/X)PHASE = 0, single phase frame
- (R/X)FRLEN2 = X
- (R/X)WDLEN1 = 101b, 32-bit words.

In this case, one 32-bit data word is transferred to and from the McBSP by the CPU or DMA. Thus, one read of DRR and one write of DXR is necessary for each frame. This results in only one-fourth the number of transfers compared to the previous case. This manipulation reduces the percentage of bus time required for serial port data movement.

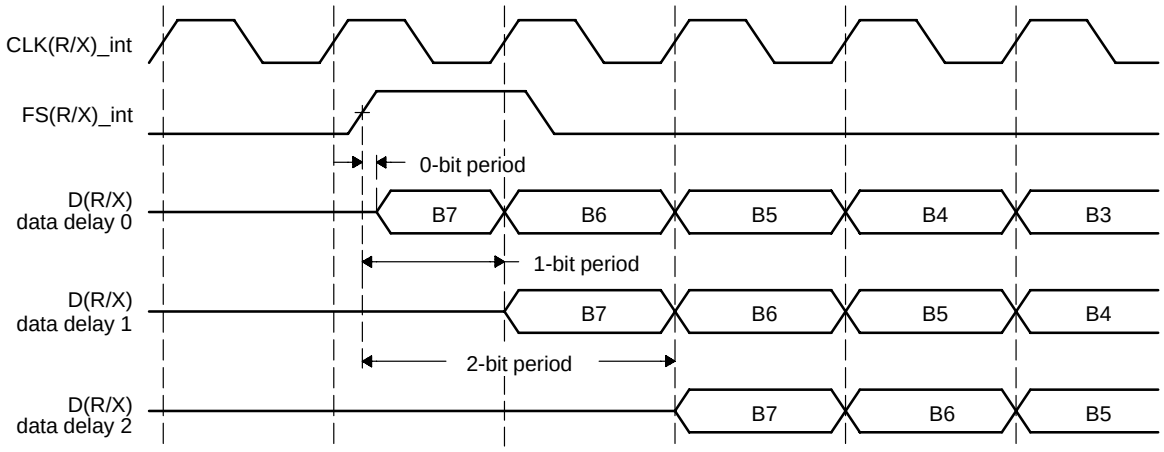
Figure 8–10. Single Phase frame of One 32-Bit Word



8.3.4.6 Data Delay: (R/X)DATDLY

The start of a frame is defined by the first clock cycle in which frame synchronization is found to be active. The beginning of actual data reception or transmission with respect to the start of the frame can be delayed if required. This delay is called data delay. RDATDLY and XDATDLY specify the data delay for reception and transmission, respectively. The range of programmable data delay is zero to two bit-clocks ((R/X)DATDLY = 00b –10b) as described in Table 8–6 and shown in Figure 8–11. Typically 1-bit delay is selected as data often follows a one-cycle active frame sync pulse.

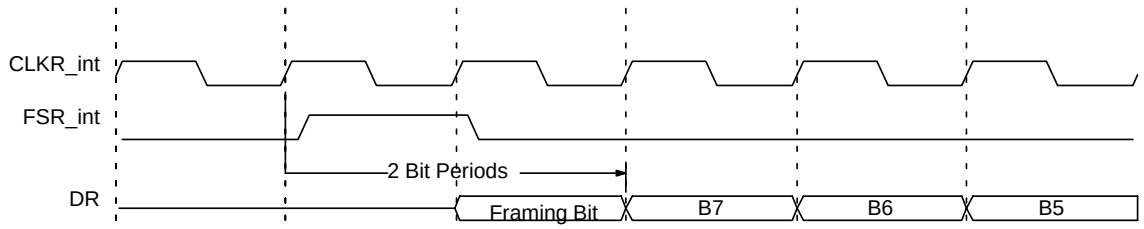
Figure 8–11. Data Delay



Normally, frame sync pulse is detected or sampled with respect to an edge of serial clock CLK(R/X)_int (See subsection 8.3.4.1). Thus, on the following cycle or later (depending on data delay value), data may be received or transmitted. However, in the case of zero-bit data delay, the data must be ready for reception and/or transmission on the same serial clock cycle. For reception, this problem is solved as receive data is sampled on the first falling edge of CLKR_int where an active high FSR_int is detected. However, data transmission must begin on the rising edge of CLKX_int clock that generated the frame synchronization. Therefore, the first data bit is assumed to be present in the XSR and thus DX. The transmitter then asynchronously detects the frame synchronization, (FSX_int), going active high, and immediately starts driving the first bit to be transmitted on the DX pin.

Another common mode is a data delay of two. This configuration allows the serial port to interface to different types of T1 framing devices where the data stream is preceded by a framing bit. During reception of such a stream with a data delay of two bits (framing bit appears after one-bit delay and data appears after 2-bit delay), the serial port essentially discards the framing bit from the data stream as shown in Figure 8–12. In transmission, by delaying the first transfer bit, the serial port essentially inserts a blank period (high impedance period) where the framing bit should be. Here, it is expected that the framing device inserts its own framing bit or that the framing bit is generated by another device.

Figure 8–12. Two-Bit Data Delay Used to Discard Framing Bit

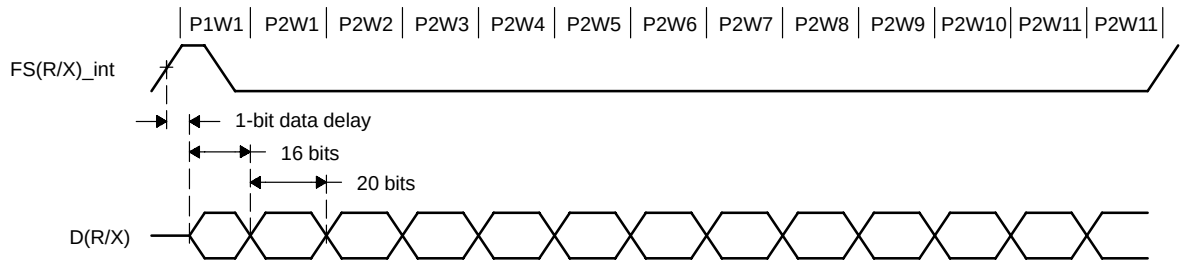


8.3.4.7 Multi-Phase Frame Example: AC97

Figure 8–13 shows an example of the Audio Codec ‘97 (AC97) standard which uses the dual phase frame feature. The first phase consists of a single 16-bit word. The second phase consists of 12 20-bit words. The phases are configured as follows:

- (R/X)PHASE = 1b, dual phase frame
- (R/X)FRLEN1 = 0b, 1 word per frame in phase 1
- (R/X)WDLEN1 = 010b, 16-bits per word in phase 1
- (R/X)FRLEN2 = 0001011b, 12 words per frame in phase 2
- (R/X)WDLEN2 = 011b, 20-bits per word in phase 2
- CLK(R/X)P = 0, receive data sampled on falling edge of CLKR_int; transmit data clocked on rising edge of CLKX_int.
- FS(R/X)P = 0, active high frame sync signals
- (R/X)DATDLY = 01b, data delay of one bit-clock

Figure 8–13. AC97 Dual Phase Frame Format†

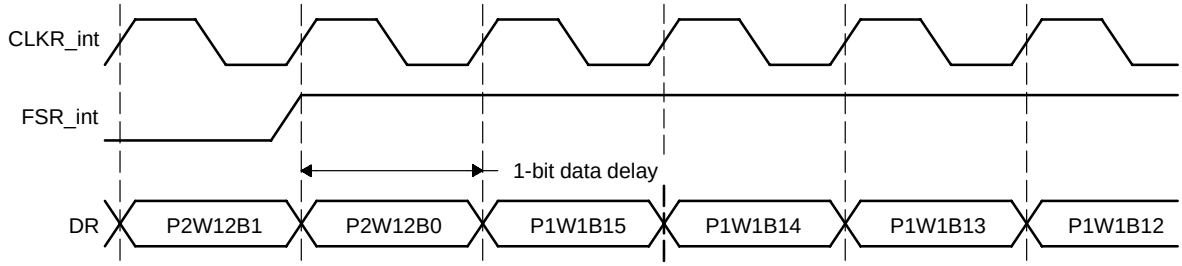


† PxWy denotes Phase x and Word y

Figure 8–13 shows the timing of AC97 near frame synchronization. First notice that the frame sync pulse itself overlaps the first word. In McBSP operation, the inactive to active transition of the frame synchronization signal actually indicates frame synchronization. For this reason, frame synchronization may be high an arbitrary number of bit clocks. Only after the frame synchronization is recognized to have gone inactive and then active again is the next frame synchronization recognized.

Also notice in Figure 8–14, there is one-bit data delay. Notice that regardless of the data delay, transmission can occur without gaps. The last bit of the previous (last) word in phase 2 is immediately followed by the first data bit of the first word in phase 1 of the next data frame.

Figure 8–14. AC97 Example Bit Timing Near Frame Synchronization†



† PxWyBz denotes Phase x, Word y, Bit z

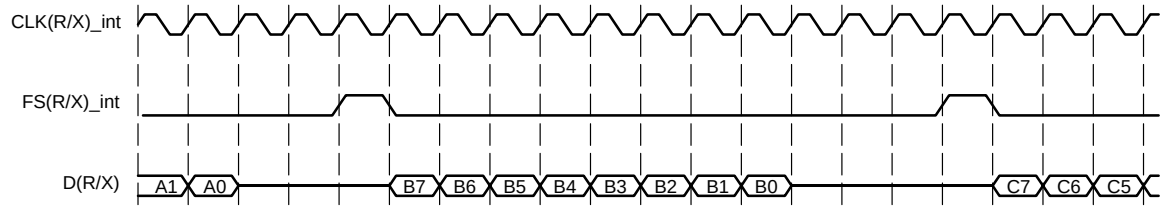
8.3.5 McBSP Standard Operation

During a serial transfer, there are typically periods of serial port inactivity between packets or transfers. The receive and transmit frame synchronization pulse occurs for every serial transfer. When the McBSP is not in reset state and has been configured for the desired operation, a serial transfer can be initiated by programming (R/X)PHASE = 0 for a single phase frame with required number of words programmed in (R/X)FRLEN1. The number of words can range from 1 to 128 ((R/X)FRLEN1 = 0x0 to 0x7F). The required serial word length is set in the (R/X)WDLEN1 field in the (R/X)CR. If dual phase is required for the transfer, RPHASE = 1, each (R/X)FRLEN(1/2) can be set to any value between 0x0 to 0x7F which represents 1 to 128 words.

Figure 8–15 shows an example of a single phase data frame comprising one 8-bit word. Since the transfer is configured for one data bit delay, the data on the DX and DR pins are available one bit clock after FS(R/X) goes active. This figure as well as all others in this section make the following assumptions:

- (R/X)PHASE = 0, single phase frame
- (R/X)FRLEN1 = 0b, 1 word per frame
- (R/X)WDLEN1 = 000b, 8-bit word
- (R/X)FRLEN2 = (R/X)WDLEN2 = X, don’t care
- CLK(R/X)P = 0, receive data clocked on falling edge; transmit data clocked on rising edge
- FS(R/X)P = 0, active high frame sync signals
- (R/X)DATDLY = 01b, one-bit data delay

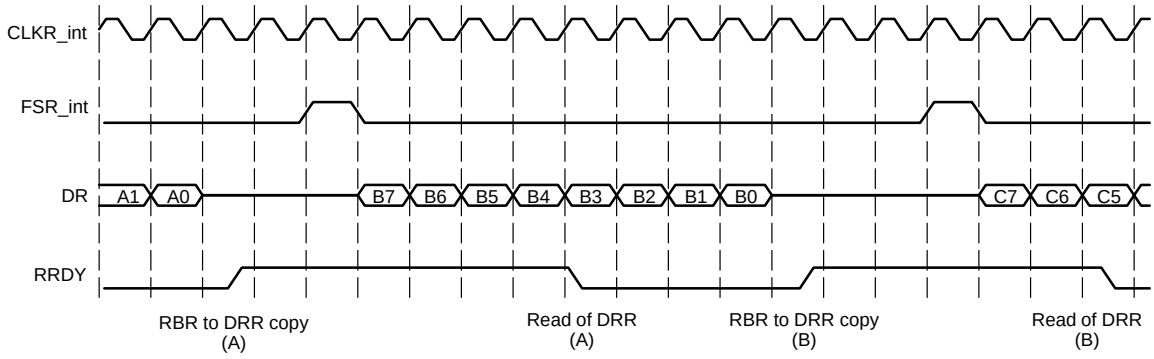
Figure 8–15. McBSP Standard Operation



8.3.5.1 Receive Operation

Figure 8–16 shows an example of serial reception. Once receive frame synchronization (FSR_int) transitions to its active state, it is detected on the first falling edge of CLKR_int of the receiver. The data on the DR pin is then shifted into the receive shift register (RSR) after the appropriate data delay as set by RDATDLY. The contents of RSR is copied to RBR at the end of every word on the rising edge of clock provided RBR is not full with the previous data. After this, an RBR-to-DRR copy activates the RRDY status bit to 1 on the following falling edge of CLKR_int. This indicates that the receive data register (DRR) is ready with the data to be read by the CPU or DMA. RRDY is deactivated when the DRR is read by the CPU or DMA.

Figure 8–16. Receive Operation

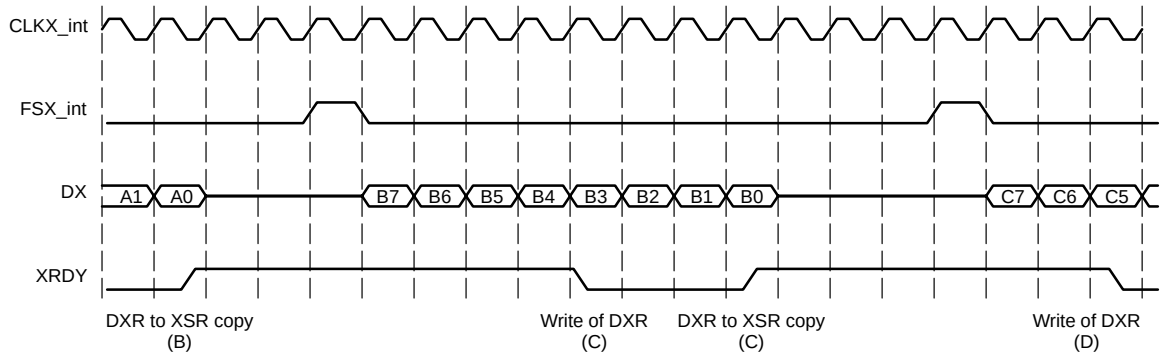


8.3.5.2 Transmit Operation

Once transmit frame synchronization occurs, the value in the transmit shift register, XSR, is shifted out and driven on the DX pin after the appropriate data delay as set by XDATDLY. XRDY is activated on every DXR-to-XSR copy on the following falling edge of CLKX_int, indicating that the data transmit register (DXR) is written with the next data to be transmitted. XRDY is deactivated when the DXR is written by the CPU or DMA. Figure 8–17 shows an example

serial transmission. See subsection 8.3.7.4 for transmit operation when transmitter is pulled out of reset ($\overline{XRST}$ = 1).

Figure 8–17. Transmit Operation



8.3.5.3 Maximum Packet Frequency

The total number of words in a frame, whether single phase or dual phase, may be called the serial transfer packet. The packet frequency is determined by the period between frame synchronization signals:

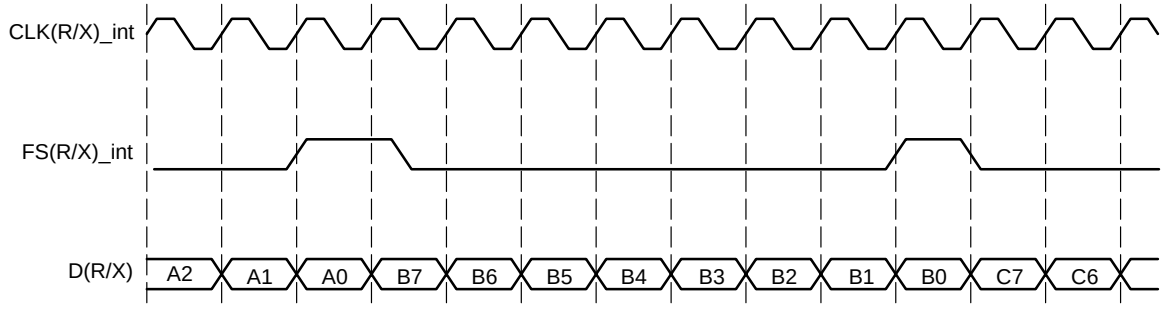
$$\text{Packet frequency} = \frac{\text{Bit-clock Frequency}}{\text{Number of Bit Clocks Between Frame Sync Signals}}$$

The packet frequency may be increased by decreasing the distance between frame synchronization signals in bit clocks (limited only by the number of bits per packet). As the packet transmit frequency is increased, the inactivity period between the data packets for adjacent transfers decreases to zero. The minimum distance between frame synchronization is the number of bits transferred per packet. This distance also defines the maximum packet frequency:

$$\text{Maximum Packet frequency} = \frac{\text{Bit-clock Frequency}}{\text{Number of Bits Per Frame or Packet}}$$

Figure 8–18 shows the McBSP operating at maximum packet frequency. At maximum packet frequency, the data bits in consecutive packets are transmitted contiguously with no inactivity between bits. If there is a one-bit data delay as shown, the frame synchronization pulse overlaps the last bit transmitted in the previous packet.

Figure 8–18. Maximum Packet Frequency Receive



Effectively, this permits a continuous stream of data, and, thus the frame synchronization pulses are essentially redundant. Theoretically, then, only an initial frame synchronization pulse is required to initiate the multi-packet transfer. The McBSP does support operation of the serial port in this fashion by ignoring the successive frame sync pulses. Data is clocked in to the receiver or clocked out of the transmitter on every clock. The frame ignore bit, (R/X)FIG, in the (R/X)CR can be programmed to ignore the successive frame sync pulses. This is explained in section 8.3.6.1.

Note:

Note that for (R/X)DATDLY = 0, the first bit of data transmitted is asynchronous to CLKX_int.

8.3.6 Frame Synchronization Ignore

The McBSP can be configured to ignore transmit and receive frame synchronization pulses. The (R/X)FIG bit in the (R/X)CR can be programmed to zero and not ignore frame sync pulses, or, be set to one and ignore frame sync pulses. This way the user can use (R/X)FIG bit to either pack data or ignore unexpected frame sync pulses. Data packing is explained in subsection 8.3.6.1, and McBSP operation on unexpected frame sync pulses in subsection 8.3.6.2.

8.3.6.1 Data Packing using Frame Sync Ignore Bits

Subsection 8.3.4.5 describes one method of changing the word length and frame length to simulate 32-bit serial word transfers, thus requiring much less bus bandwidth. This example worked when there were multiple words per frame. Now consider the case of the McBSP operating at maximum packet frequency as shown in Figure 8–19. Here, each frame only has a single 8-bit word. This stream takes one read and one write transfer for each 8-bit word. Figure 8–20 shows the McBSP configured to treat this stream as a continuous stream of 32-bit words. In this example (R/X)FIG is set to 1 to ignore subsequent frames after the first. Here, only one read and one write transfer is needed every 32-bits. This configuration effectively reduces the required bus bandwidth to one-fourth.

Figure 8–19. Maximum Packet Frequency Operation with 8-Bit Data

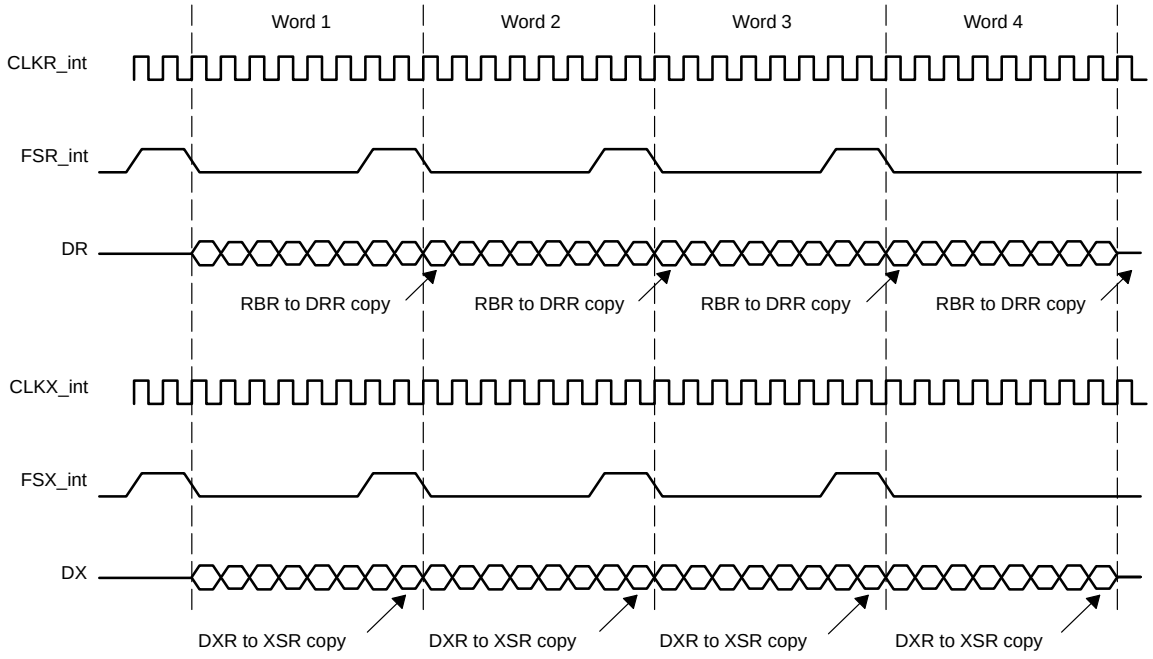
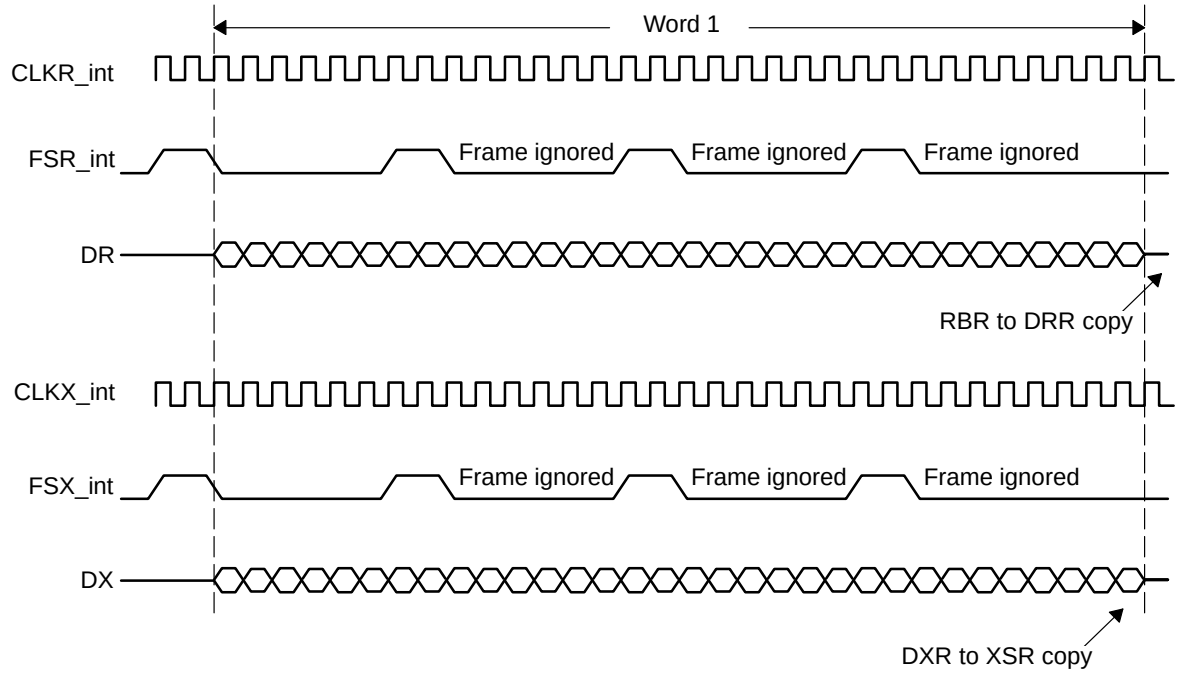


Figure 8–20. Data Packing at Maximum Packet Frequency with (R/X)FIG = 1



8.3.6.2 Frame Sync Ignore and Unexpected Frame Sync Pulses

The previous subsection explains how frame ignore bits can be used to pack data and efficiently use the bus bandwidth. (R/X)FIG bit can also be used to ignore unexpected frame sync pulses. Unexpected frame sync pulses are those that occur at a time when they are not expected. Thus, any frame sync pulse which occurs one or more bit clocks earlier than the programmed data delay ((R/X)DATDLY) is deemed as unexpected. Setting the frame ignore bits to one causes the serial port to ignore these unexpected frame sync signals.

In reception, if not ignored (RFIG = 0), an unexpected FSR pulse will discard the contents of RSR in favor of the new incoming data. Therefore if RFIG = 0, an unexpected frame synchronization pulse aborts the current data transfer, sets RSYNCERR in the SPCR to 1, and begins the reception of a new data word. See subsection 8.3.7.2 for further details. When RFIG = 1, reception continues, ignoring the unexpected frame sync pulses.

If (R/X)FIG is set to zero, these frame sync pulses are not ignored. In transmission, if not ignored (XFIG = 0), an unexpected FSX pulse will abort the ongoing transmission, set the XSYNCERR in the SPCR to 1, and re-initiate transmission of the current word that was aborted. See subsection 8.3.7.5 for further details. When XFIG = 1, normal transmission continues with unexpected frame sync signals ignored.

Figure 8–21. Unexpected Frame Synchronization with (R/X)FIG = 0

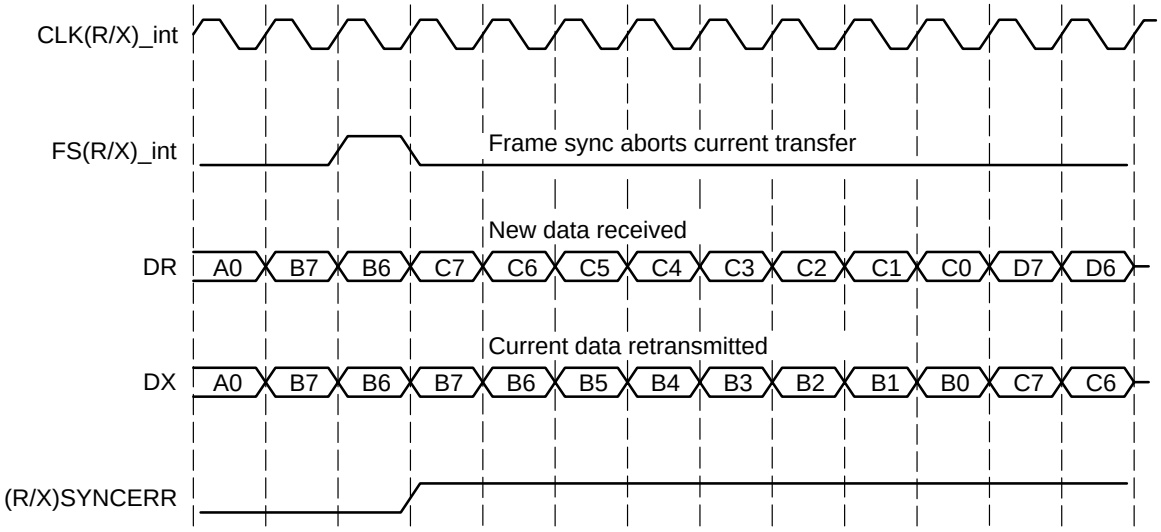
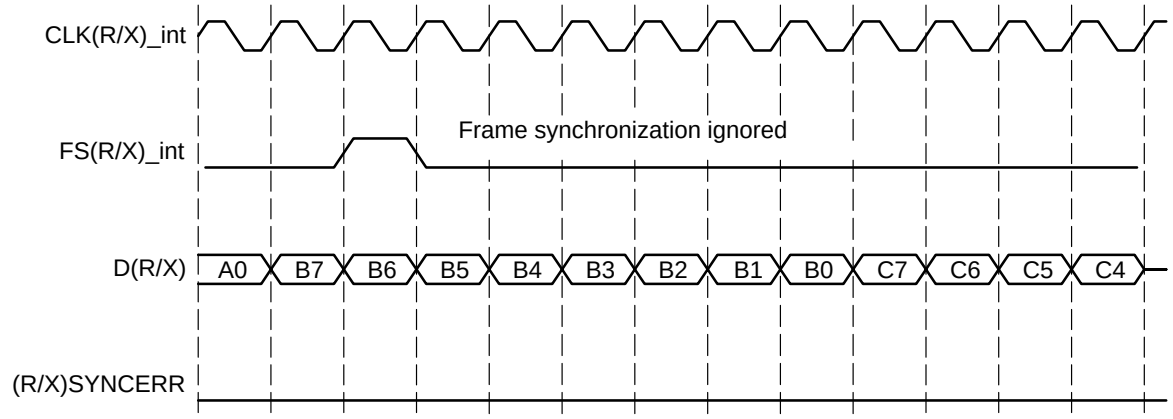


Figure 8–21 shows an example wherein word B is interrupted by an unexpected frame sync pulse when (R/X)FIG = 0. In the case of reception, the reception of B is aborted (B is lost), and a new data word (C in this example) is received after the appropriate data delay. This condition is a receive synchronization error and thus sets the RSYNCERR field. However, for transmission, the transmission of B is aborted, and the same data (B) is re-transmitted after the appropriate data delay. This condition is a transmit synchronization error and thus sets the XSYNCERR field. Synchronization errors are discussed in subsections 8.3.7.2 and 8.3.7.5. In contrast, Figure 8–22 shows McBSP operation when unexpected frame synchronization signals are ignored by setting (R/X)FIG = 1. Here, the transfer of word B is not affected by an unexpected frame synchronization.

Figure 8–22. Unexpected Frame Synchronization with (R/X)FIG = 1



8.3.7 Serial Port Exception Conditions

There are five serial port events that may constitute a system error:

1) **Receive Overrun (RFULL = 1)**. This occurs when DRR (with data A) has not been read since the last RBR-to-DRR transfer, therefore a new word (data B) in RBR has not been transferred to DRR, and RSR is now full with another new word (data C) shifted in from DR. Therefore, RFULL indicates an error condition wherein any new data that may arrive at this time on DR will replace the contents in RSR, and thus previous word (data C) is lost. RSR continues to be overwritten as long as new data arrives on the DR pin and DRR is not read.

2) **Unexpected Receive Frame Synchronization (RSYNCERR = 1)**. This can occur during reception when RFIG = 0 and an unexpected frame sync pulse occurs. An unexpected frame sync pulse is defined as that which occurs RDATDLY minus 1 or more bit-clocks earlier than the first bit of the next associated word. This causes the current data reception to abort and restart. If new data has been copied into the RBR from RSR since the last RBR-to-DRR copy, this new data in RBR will be lost because no RBR-to-DRR copy occurs as the reception has been restarted.

3) **Transmit Data Overwrite**. Here the user overwrites data in the DXR before it is copied to the XSR. The data previously in the DXR is never transferred on DX since it never got copied to the XSR.

4) **Transmit Empty ($\overline{\text{XEMPTY}}$ = 0)**. If a new frame synchronization signal arrives before new data is loaded into the DXR, the old data in the DXR will be sent again. This will continue for every new frame sync signal that arrives on the FSX pin until the DXR is loaded with new data.

5) **Unexpected Transmit Frame Synchronization (XSYNCERR = 1)**. This can occur during transmission when XFIG = 0 and an unexpected frame sync pulse occurs. Again, an unexpected frame sync pulse is defined as that which occurs XDATDLY minus 1 or more bit-clocks earlier than the first bit of the next associated word. This causes the current data transmission to abort and restart the current transfer. If new data has been written to the DXR since the last DXR-to-XSR copy, the current value in the XSR will be lost.

These events are described in more detail in the following subsections.

8.3.7.1 Reception with Overrun: RFULL

RFULL = 1 in the SPCR indicates that the receiver has experienced overrun and is in an error condition. RFULL is set when:

1) DRR has not been read since the last RBR-to-DRR transfer.
2) RBR is full, and a RBR to DRR copy has not occurred, and
3) RSR is full, and a RSR to RBR transfer has not occurred.

The data arriving on DR is continuously shifted into RSR. Once a complete word is shifted into RSR, an RSR-to-RBR transfer can occur only if an RBR-to-DRR copy is complete. Therefore, if DRR has not been read by the CPU or DMA since the last RBR-to-DRR transfer (RRDY = 1), an RBR-to-DRR copy will not take place until RRDY = 0. This prevents an RSR to RBR copy. At this time, new data arriving on DR pin is shifted into RSR and the previous contents of RSR is lost. This data loss occurs because completion of reception of a serial word triggers an RBR-to-DRR transfer only when RRDY = 0. Note that after the receive portion starts running from reset, a minimum of three words must be received before RFULL is set because there was no last RBR-to-DRR transfer before the first word.

This data loss can be avoided if DRR is read no later than two and a half cycles before the end of (third) next word (data C) in RSR.

Any one of the following events clears the RFULL bit to 0 and allows subsequent transfers to be read properly:

1) Reading DRR
2) Resetting the receiver ($\overline{\text{RRST}}$ = 0) or the device.

Another frame synchronization is required to restart the receiver.

Figure 8–23 shows the receive overrun condition. Because serial word A is not read before the reception of serial word B is complete, B is not transferred to DRR yet. Now, another new word C arrives and RSR is full with this data. DRR is finally read, but not earlier than two and one half cycles before the end of word C. Therefore, new data D overwrites the previous word C in RSR. If RFULL is still set after the DRR is read, the next word can overwrite D, if DRR is not read in time.

Figure 8–23. Serial Port Receive Overrun

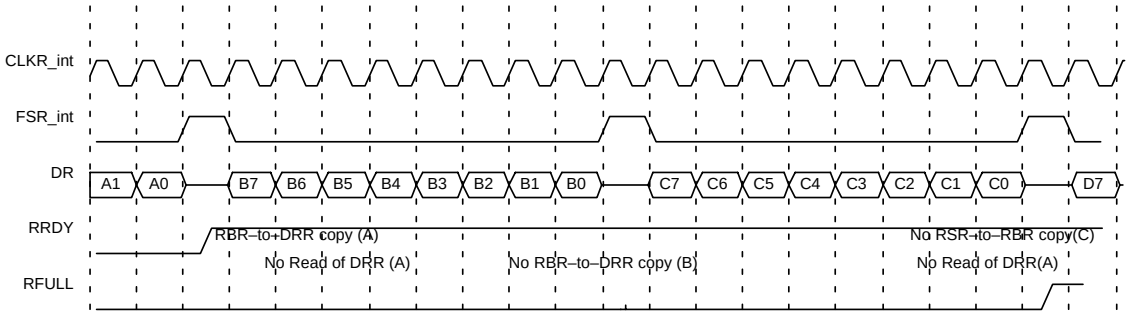
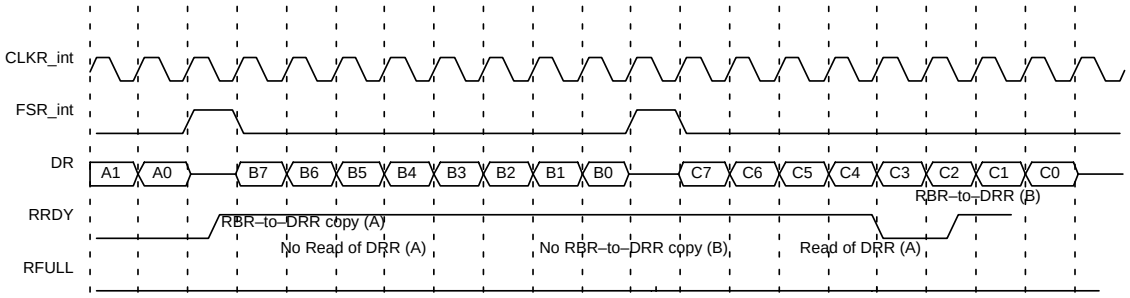


Figure 8–24 shows the case where RFULL is set, but the overrun condition is averted by reading the contents of DRR at least two and a half cycles before the next serial word C is completely shifted into RSR. This ensures that a RBR-to-DRR copy of data B occurs before the next serial word (C) is transferred from RSR to RBR.

Figure 8–24. Serial Port Receive Overrun Avoided



8.3.7.2 Unexpected Receive Frame Synchronization: RSYNCERR

Figure 8–25 shows the decision tree that the receiver uses to handle all incoming frame synchronization pulses. The diagram assumes that the receiver has been activated, $\overline{\text{RRST}}$ = 1. Unexpected frame sync pulses can originate from an external source or from the internal sample rate generator. Any one of four cases can occur:

- Case 1 : Unexpected FSR_int pulses with RFIG = 1. This case is discussed in 8.3.6.2 and shown in Figure 8–22. Here, receive frame sync pulses are ignored and the reception continues.

- Case 2: Normal serial port reception. Note that there are three possible reasons why a receive might NOT be in progress:

  1) This FSR is the first after $\overline{\text{RRST}}$ = 1.

  2) This FSR is the first after DRR is read clearing a RFULL condition.

  3) The serial port is in the inter-packet intervals. The programmed data delay (RDATDLY) for reception may start during these inter-packet intervals for the first bit of the next word to be received. Thus, at maximum packet frequency, frame synchronization can still be received RDATDLY bit clocks before the first bit of the associated word. Alternatively, unexpected frame sync pulses are detected when they occur at or before RDATDLY minus 1 or more bit clocks before the last bit of the previous word is received on DR pin.

  For this case, reception continues normally since these are not unexpected frame sync pulses.

- Case 3: Unexpected receive frame synchronization with RFIG = 0 (unexpected frame not ignored). This case was shown in Figure 8–21 for maximum packet frequency. Figure 8–26 shows this case during normal operation of the serial port with inter-packet intervals. In both cases, RSYNCERR bit in the SPCR is set. RSYNCERR can be cleared only by receiver reset or by the user writing a 0 to this bit in the SPCR. Note that if RINTM = 11b in the SPCR, RSYNCERR drives the receive interrupt (RINT) to the CPU.

Note:

Note that the RSYNCERR bit in the SPCR is a read/write bit. Therefore writing a 1 to it, sets the error condition. Typically, writing a 0 is expected.

Figure 8–25. Response to Receive Frame Synchronization Pulse

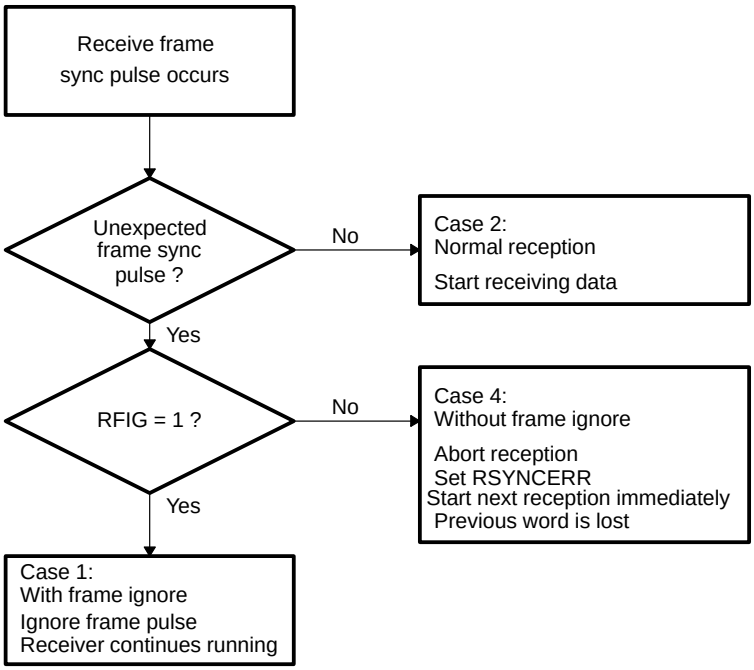
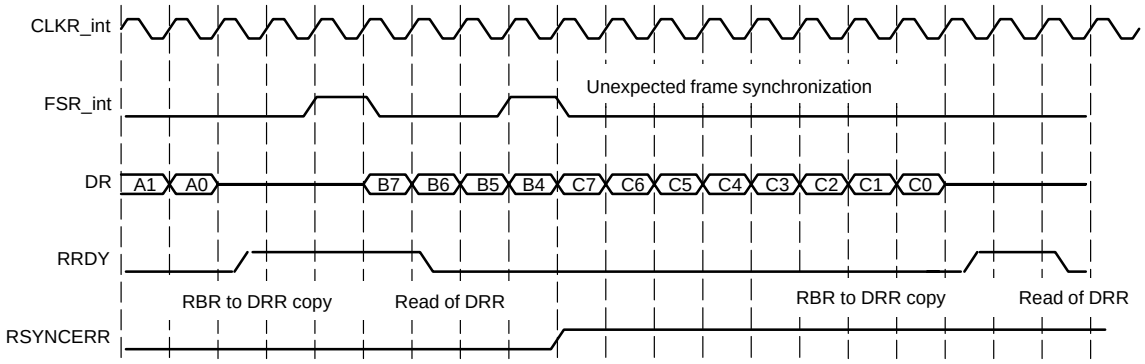


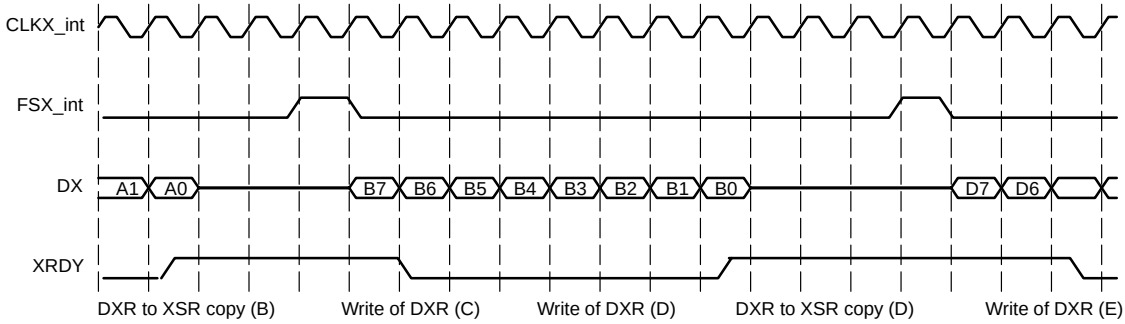
Figure 8–26. Unexpected Receive Synchronization Pulse



8.3.7.3 Transmit with Data Overwrite

Figure 8–27 depicts what happens if the data in DXR is overwritten before being transmitted. Initially, the programmer loaded the DXR with data C. A subsequent write to the DXR overwrites C with D before it is copied to the XSR. Thus, C is never transmitted on DX. The CPU can avoid overwriting data by polling XRDY before writing to DXR or by waiting for an XINT programmed to be triggered by XRDY (XINTM = 00b). The DMA can avoid overwriting by write synchronizing data transfers with XEVT.

Figure 8–27. Transmit with Data Overwrite



8.3.7.4 Transmit Empty: $\overline{\text{XEMPTY}}$

$\overline{\text{XEMPTY}}$ indicates whether the transmitter has experienced under-flow. Any of the following conditions causes $\overline{\text{XEMPTY}}$ to become active ($\overline{\text{XEMPTY}}$ = 0):

1) Under-flow during transmission occurs when DXR has not been loaded since the last DXR-to-XSR copy, and all bits of the data word in the XSR have been shifted out on DX.

2) The transmitter is reset ($\overline{\text{XRST}}$ = 0 or device is reset) and then restarted.

During underflow condition, the transmitter continues to transmit the old data in DXR until a new word is loaded into DXR by the CPU or DMA. $\overline{\text{XEMPTY}}$ is deactivated ($\overline{\text{XEMPTY}}$ = 1) after DXR is loaded by either the CPU or DMA. In the case of internal frame generation, the transmitter re-generates a single FSX_int initiated by a DXR-to-XSR copy (FSXM = 1 in the PCR and FSGM = 0 in SRGR). Otherwise, the transmitter waits for the next frame synchronization.

When the transmitter is taken out of reset ($\overline{\text{XRST}}$ = 1), it is in a transmit ready (XRDY = 1) and transmit empty ($\overline{\text{XEMPTY}}$ = 0) condition. If DXR is loaded by the CPU or DMA before FSX_int goes active high, a valid DXR-to-XSR transfer occurs. This allows for the first word of the first frame to be valid even before the transmit frame sync pulse is generated or detected. Alternatively, if a transmit frame sync is detected before DXR is loaded, zeros will be output on DX.

Figure 8–28 depicts a transmit under-flow condition. After B is transmitted, the programmer fails to reload the DXR before the subsequent frame synchronization. Thus, B is again transmitted on DX. Figure 8–29 shows the case of writing to DXR just before a transmit under-flow condition that would otherwise occur. After B is transmitted, C is written to DXR before the next transmit frame sync pulse occurs so that C is successfully transmitted on DX, averting a transmit empty condition.

Figure 8–28. Transmit Empty

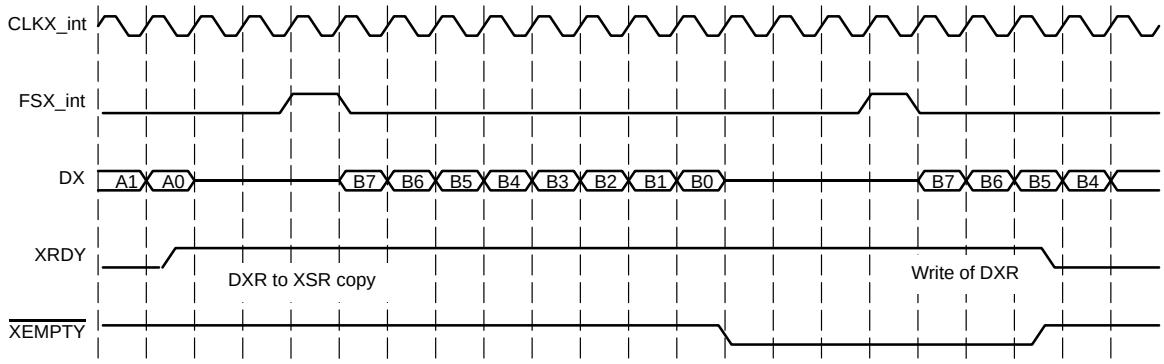
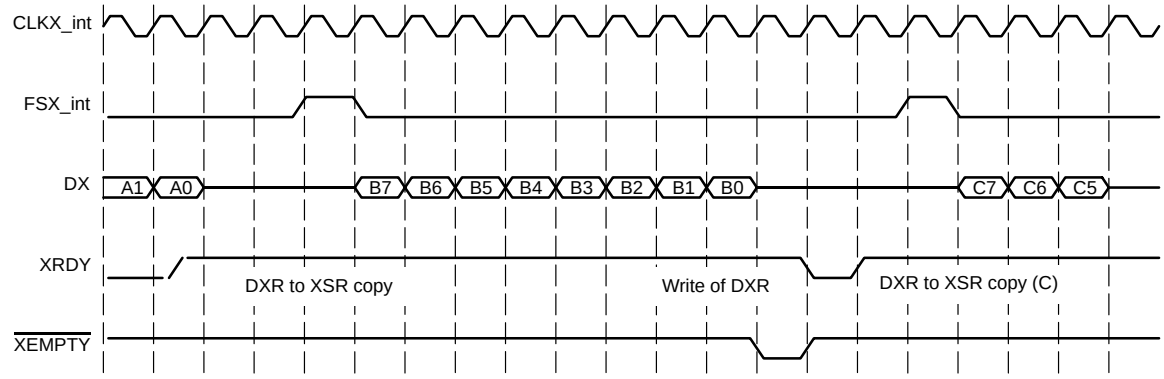


Figure 8–29. Transmit Empty Avoided



8.3.7.5 Unexpected Transmit Frame Synchronization: XSYNCERR

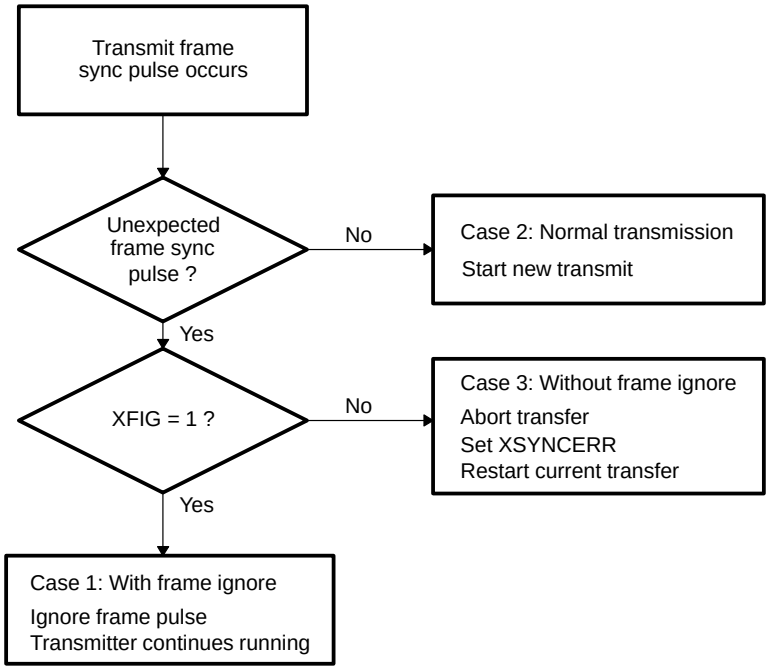
Figure 8–25 shows the decision tree that the transmitter uses to handle all incoming frame synchronization signals. The diagram assumes that the transmitter has been started, $\overline{\text{XRST}}$ = 1. Any one of three cases can occur:

- ❑ Case 1: Unexpected FSX_int pulses with RXFIG = 1. This case is discussed in 8.3.6.2 and shown in Figure 8–22.
- ❑ Case 2: Normal serial port transmission is discussed in subsection 8.3.5.3. Note that there are two possible reasons why a transmit might NOT be in progress:

  1) This FSX_int pulse is the first after $\overline{\text{XRST}}$ = 1.

  2) The serial port is in the inter-packet intervals. The programmed data delay (XDATDLY) may start during these inter-packet intervals before the first bit of the next word is transmitted. Thus, if operating at maximum packet frequency, frame synchronization can still be received XDATDLY bit clocks before the first bit of the associated word.

Figure 8–30. Response to Transmit Frame Synchronization

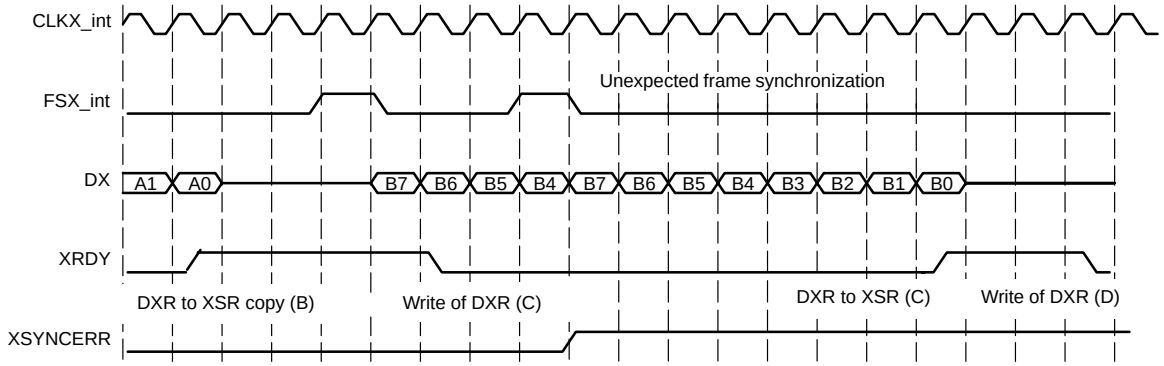


- Case 3: Unexpected transmit frame synchronization with XFIG = 0. The case for subsequent frame synchronization with XFIG = 0 at maximum packet frequency is shown in Figure 8–21. Figure 8–31 shows the case for normal operation of the serial port with inter-packet intervals. In both cases, XSYNCERR bit in the SPCR is set. XSYNCERR can only be cleared by transmitter reset or by the user writing a 0 to this bit in the SPCR. Note that if XINTM = 11b in the SPCR, XSYNCERR drives the receive interrupt (XINT) to the CPU.

Note:

Note that the XSYNCERR bit in the SPCR is a read/write bit. Therefore writing a 1 to it, sets the error condition. Typically, writing a 0 is expected.

Figure 8–31. Unexpected Transmit Frame Synchronization Pulse



8.3.8 Receive Data Justification and Sign-Extension: RJUST

RJUST in the SPCR selects whether data in the RBR is right or left justified (with respect to the MSB) in the DRR. If right-justification is selected RJUST further selects whether the data is sign-extended or zero-filled. Table 8–11 shows the effect various modes of RJUST have on an example 12-bit receive data value 0xABC.

Table 8–11. Use of RJUST Field with 12-Bit Example Data 0xABC

RJUST	Justification	Extension	Value in DRR
00	right	zero-fill MSBs	0x00000ABC
01	right	sign-extend MSBs	0xFFFFFABC
10	left	zero-fill LSBs	0xABC00000
11	reserved	reserved	reserved

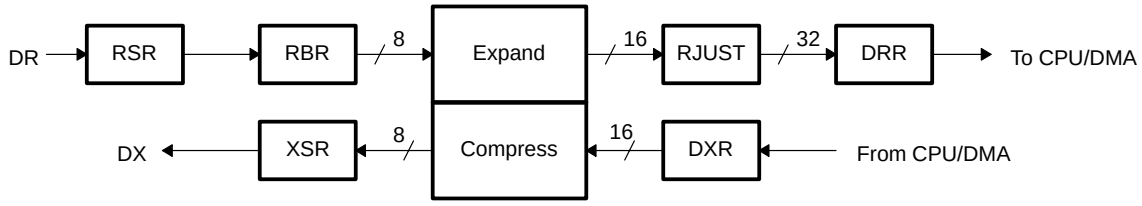
8.4 μ-LAW/A-LAW Companding Hardware Operation

Companding (COMpress and exPAND) hardware allows compression and expansion of data in either μ-law or A-law format. The companding standard employed in the United States and Japan is μ-law. The European companding standard is referred as A-law. The specification for μ-law and A-law log PCM is part of the CCITT G.711 recommendation. A-law and μ-law allow 13-bits and 14-bits of dynamic range, respectively. Any values outside this range will be set to the most positive or most negative value. Thus, for companding to work best, the data transferred to and from the McBSP via the CPU or DMA must be at least 16-bit wide data.

The μ-law and A-law formats encode this data into 8-bit code words. Thus, as companded data is always 8-bit wide, the appropriate (R/X)WDLEN(1/2) must be set to 0, indicating 8-bit wide serial data stream. If either phase of the frame does not have an 8-bit word length, companding continues as if the word length is 8-bits.

When companding is used, transmit data is encoded according to the specified companding law, and receive data is decoded to 2's complement format. Companding is enabled and the desired format selected by appropriately setting (R/X)COMPAND in the (R/X)CR as shown in Table 8–6. Compression occurs during the process of copying data from DXR to XSR and from RBR to DRR as shown in Figure 8–32.

Figure 8–32. Companding Flow



For transmit data to be compressed, it should be a 16-bit left justified data, say LAW16. The value can be either 13- or 14-bits depending on the companding law as shown in Figure 8–33. This 16-bit transmit data is aligned in DXR as shown in Table 8–12.

Figure 8–33. Companding Data Formats

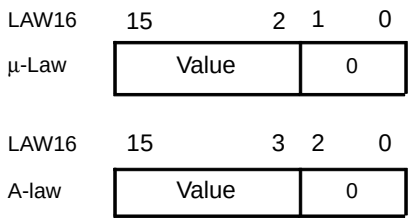


Table 8–12. Transmit Data Companding Format

DXR	
31 16	15 0
Don't Care	LAW16

For reception, the 8-bit compressed data in RBR is expanded to a left-justified 16-bit data, LAW16. This can be further justified to a 32-bit data by programming the RJUST field in the SPCR as shown in Table 8–13.

Table 8–13. Justification of Expanded Data (LAW16)

	DRR	
RJUST	31 16	15 0
00	0	LAW16
01	sign	LAW16
10	LAW16	0
11	reserved	

8.4.1 Companding Internal Data

If the McBSP is otherwise unused, the companding hardware can compand internal data. This hardware can be used to:

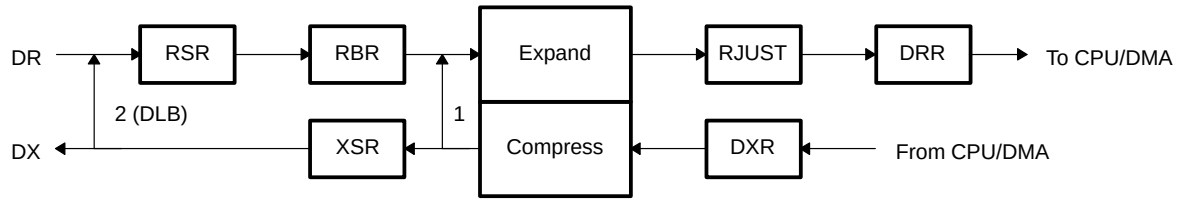
- Convert linear to the appropriate μ-law or A-law format.
- Convert μ-law or A-law to the linear format.
- To observe the quantization effects in companding by transmitting linear data, and compressing and re-expanding this data. This is only useful if both XCOMPAND and RCOMPAND enable the same companding format.

Figure 8–34 shows two methods by which the McBSP can compand internal data.

1) When both the transmit and receive sections of the serial port are reset, the DRR and DXR are internally connected through the companding logic. Values from the DXR are compressed as selected by XCOMPAND and then expanded as selected by RCOMPAND. Note that RRDY and XRDY bits will not be set. However, data is available in DRR within four CPU clocks after being written to DXR. The advantage of this method is its speed. The disadvantage is that there is no synchronization available to the CPU and DMA to control the flow.

2) The McBSP is enabled in digital loop back mode with companding appropriately enabled by RCOMPAND and XCOMPAND. Receive and transmit interrupts (RINT when RINTM = 0 and XINT when XINTM = 0) or synchronization events (REVT and XEVT) allow synchronization of the CPU or DMA to these conversions, respectively. Here, the time for this companding depends on the serial bit rate selected.

Figure 8–34. Companding of Internal Data



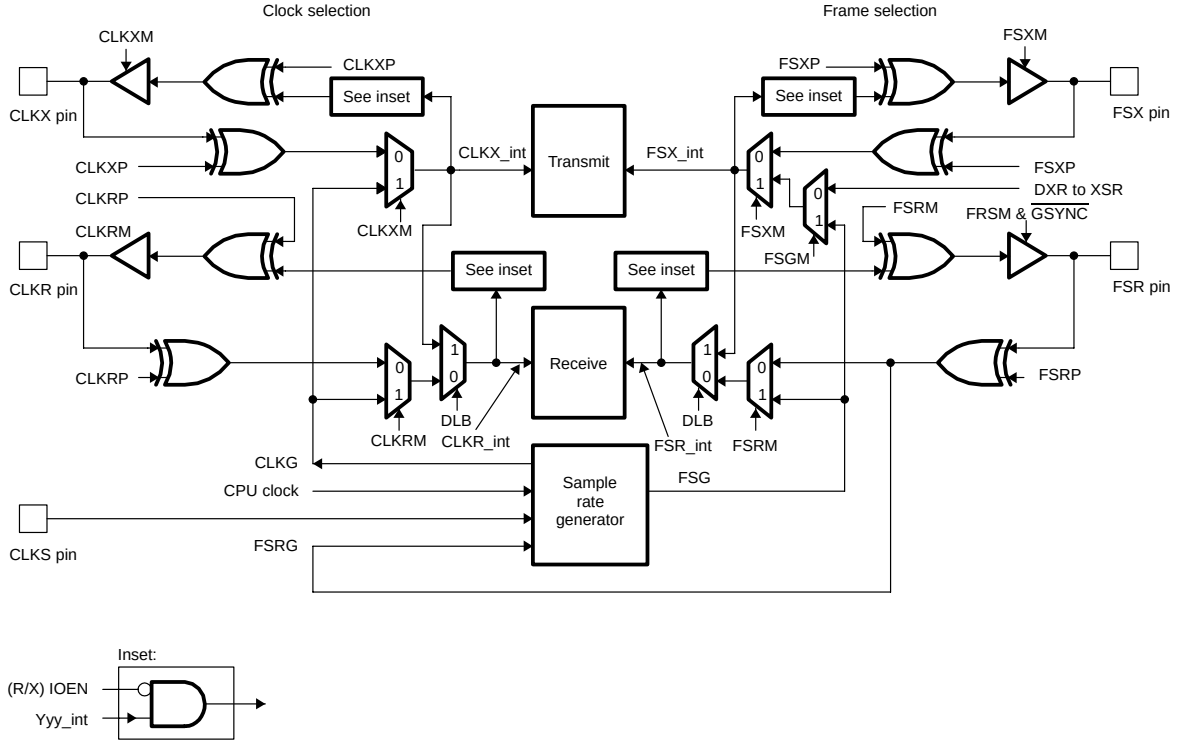
8.4.1.1 Bit Ordering

Normally, all transfers on the McBSP are sent and received with the MSB first. However, certain 8-bit data protocols (that don't use companded data) require the LSB to be transferred first. By setting the (R/X)COMPAND = 01b in the (R/X)CR, the bit ordering of 8-bit words are reversed (LSB first) before being sent to the serial port. Similar to companding, this feature is only enabled if the appropriate (R/X)WDLEN(1/2) is set to 0, indicating 8-bit words to be transferred serially.

8.5 Programmable Clock and Framing

The McBSP has several means of selecting clocking and framing for both the receiver and transmitter. Clocking and framing can be sent to both portions by the sample rate generator. Both portions can select external clocking and/or framing independently. Figure 8–35 shows a block diagram of the clock and frame selection circuitry. The features enabled by this logic are explained in the following subsections.

Figure 8–35. Clock and Frame Generation



8.5.1 Sample Rate Generator Clocking and Framing

The sample rate generator is composed of a three stage clock divider that allows programmable data clocks (CLKG) and framing signals (FSG) as shown in Figure 8–36. CLKG and FSG are McBSP internal signals that can be programmed to drive receive and/or transmit clocking (CLKR/X) and framing (FSR/X). The sample rate generator can be programmed to be driven by an internal clock source or an internal clock derived from an external clock source. The three stages compute:

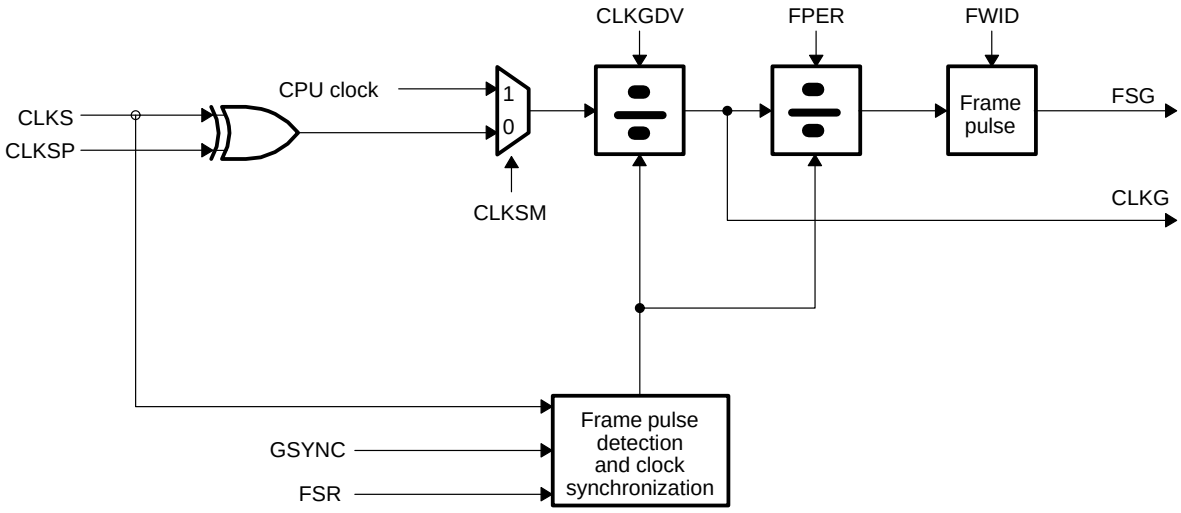
1) Clock divide down (CLKGDV): The number of input clocks per data bit clock.

2) Frame period divide down (FPER): The frame period in data bit clocks.

3) Frame width count down (FWID): The width of an active frame pulse in data bit clocks.

In addition, a frame pulse detection and clock synchronization module allows synchronization of the clock divide down with an incoming frame pulse. The operation of the sample rate generator during device reset is described in subsection 8.3.1.

Figure 8–36. Sample Rate Generator



8.5.1.1 Sample Rate Generator Register (SRGR)

The Sample Rate Generator Register (SRGR) shown in Figure 8–37 and Table 8–14 controls the operation of various features of the sample rate generator. The following subsections describe how you can configure its operation using the SRGR bit-fields.

Figure 8–37. Sample Rate Generator Register (SRGR)

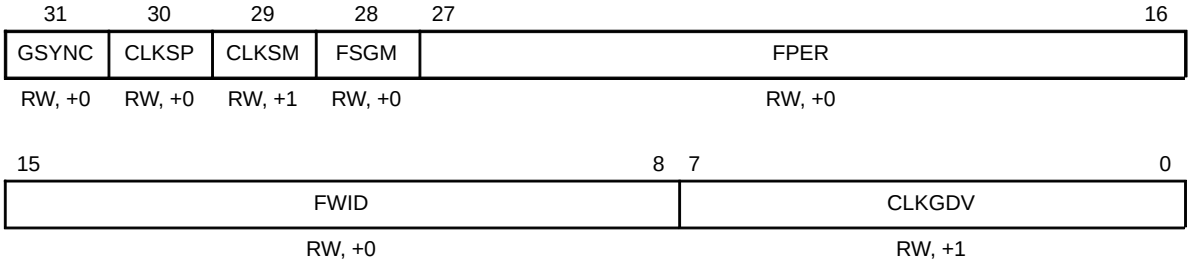


Table 8–14. Sample Rate Generator Register (SRGR) Bit-Field Summary

Name	Function	Section
GSYNC	Sample rate generator clock synchronization. Only used when the external clock (CLKS) drives the sample rate generator clock (CLKSM = 0). GSYNC = 0, the sample rate generator clock (CLKG) is free running. GSYNC = 1, the sample rate generator clock (CLKG) is running. But CLKG is re-synchronized and frame sync signal (FSG) is generated only after detecting the receive frame synchronization signal (FSR). Also, frame period, FPER, is a don't care because the period is dictated by the external frame sync pulse.	8.5.2.4
CLKSP	CLKS Polarity Clock Edge Select. Only used when the external clock CLKS drives the sample rate generator clock (CLKSM = 0). CLKSP = 0, rising edge of CLKS generates CLKG and FSG. CLKSP = 1, falling edge of CLKS generates CLKG and FSG.	8.5.2.3
CLKSM	McBSP Sample Rate Generator Clock Mode CLKSM = 0, Sample rate generator clock derived from the CLKS pin. CLKSM = 1, Default value; Sample rate generator clock derived from CPU clock.	8.5.2.2
FSGM	Sample Rate Generator Transmit frame synchronization mode. Used when FSXM = 1 in PCR. FSGM = 0, Transmit frame sync signal (FSX) generated on every DXR-to-XSR copy. FSGM = 1, Transmit frame sync signal driven by the sample rate generator frame sync signal, FSG.	8.5.3.3

Table 8–14. Sample Rate Generator Register (SRGR) Bit-Field Summary (Continued)

Name	Function	Section
FPER	Frame Period. This determines when the next frame sync signal should become active. Range: up to 2^{12} ; 1 to 4096 CLKG periods.	8.5.3.1
FWID	Frame Width. Determines the width of the frame sync pulse, FSG, during its active period. Range: up to 2^{8} ; 1 to 256 CLKG periods.	8.5.3.1
CLKGDV	Sample rate generator clock divider. This value is used as the divide-down number to generate the required sample rate generator clock frequency. Default value is 1.	8.5.2.2

8.5.1.2 Sample Rate Generator Reset Procedure

The sample rate generator reset and initialization procedure is as follows:

1) During device reset, $\overline{\text{GRST}}$ = 0. Otherwise, during normal operation, the sample rate generator can be reset with $\overline{\text{GRST}}$ = 0 in SPCR, provided CLKG and/or FSG ($\overline{\text{FRST}}$ = 1) is not used by any portion of the McBSP. If $\overline{\text{GRST}}$ is low due to device reset, CLKG is driven by a divide-by-2 CPU clock and FSG is driven inactive low. If $\overline{\text{GRST}}$ = 0 as desired by the user, CLKG and FSG are driven inactive low. If necessary, set $\overline{\text{(R/X)RST}}$ = 0.

2) Program SRGR as required. If necessary, other control registers can be written with desired values provided the respective portion (R/X) is in reset.

3) Wait two CLKSRG clocks. This is to ensure proper synchronization internally.

4) Set $\overline{\text{GRST}}$ = 1 to enable the sample rate generator.

5) Wait two CLKG bit clocks.

6) Now, the receiver and/or transmitter can be pulled out of reset ($\overline{\text{(R/X)RST}}$ = 1) if required.

7) On the next rising edge of CLKSRG, CLKG transitions to 1 and starts clocking with a frequency equal to (CPU clock / (1+CLKGDV)).

8) After the required data acquisition setup is done such as writing to DXR, $\overline{\text{FRST}}$ can be written with 1 in the SPCR, if an internally generated frame pulse is required. FSG is generated with a active high edge after the programmed number of (FPER+1) CLKG clocks have elapsed. For example, if FPER = 7, FSG is generated (if FSGM = 1) after 8 CLKG clocks.

8.5.2 Data Clock Generation

When the receive/transmit clock mode is set to 1 (CLK(R/X)M = 1), the data clocks (CLK(R/X)) are driven by the internal sample rate generator output clock, CLKG. You can select from a variety of data bit clocks independently for the receiver and transmitter. These options include:

- ❑ The input clock to the sample rate generator can be either the CPU clock or a dedicated external clock source (CLKS).
- ❑ The input clock (CPU clock or external clock CLKS) source to the sample rate generator can be divided down by a programmable value (CLKGDV) to drive CLKG.

Irrespective of the source to the sample rate generator, the rising edge of CLKSRG (see Figure 8–36) generates CLKG and FSG (also, see subsection 8.5.2.3).

8.5.2.1 Input Clock Source Mode: CLKSM

The CLKSM bit in the SRGR selects either the CPU clock (CLKSM = 1) or the external clock input (CLKSM = 0), CLKS, as the source for the sample rate generator input clock. Any divide periods are divide-downs calculated by the sample rate generator and are timed by this input clock selection.

8.5.2.2 Sample Rate Generator Data Bit Clock Rate: CLKGDV

The first divider stage generates the serial data bit clock from the input clock. This divider stage utilizes a counter that is pre-loaded by CLKGDV which contains the divide ratio value. The output of this stage is the data bit clock which is output on sample rate generator output, CLKG, and serves as the input for the second and third divider stages.

CLKG has a frequency equal to 1/(CLKGDV+1) of sample rate generator input clock. Thus, sample generator input clock frequency is divided by a value from 1–256. When CLKGDV is odd or equal to 0, the CLKG duty cycle is 50%. When CLKGDV is an even value, 2p, representing an odd divide-down, the high state duration is p+1 cycles and the low state duration is p cycles.

8.5.2.3 Bit Clock Polarity: CLKSP

External clock (CLKS) is selected to drive the sample rate generator clock divider by selecting CLKSM = 0. In this case, the CLKSP bit in the SRGR selects the edge of CLKS on which sample rate generator data bit clock (CLKG) and frame sync signal (FSG) are generated. Since the rising edge of CLKSRG (see Figure 8–36) generates CLKG and FSG, the rising edge of CLKS when CLKSP = 0 or the falling edge of CLKS when CLKSP = 1 causes the transition on the data bit-rate clock (CLKG) and FSG.

8.5.2.4 Bit Clock and Frame Synchronization

When CLKS is selected to drive the sample rate generator (CLKSM = 0), GSYNC can be used to configure the timing of CLKG relative to CLKS. GSYNC = 1 ensures that the McBSP and the external device it is communicating to, are dividing down the CLKS with the same phase relationship. If GSYNC = 0, this feature is disabled and therefore CLKG runs freely and is not re-synchronized. If GSYNC = 1, an active transition on FSR triggers a re-synchronization of CLKG and generation of FSG. CLKG always begins with a high state after synchronization. Also, FSR is always detected at the same edge of CLKS that generates CLKG, no matter how wide the FSR is. Although an external FSR is provided, FSG can still drive internal receive frame synchronization when GSYNC = 1. Note that when GSYNC = 1, FPER is a don't care because the frame period is determined by the arrival of the external frame sync pulse.

Figure 8–38. CLKG Synchronization and FSG generation when GSYNC = 1 and CLKGDV = 1

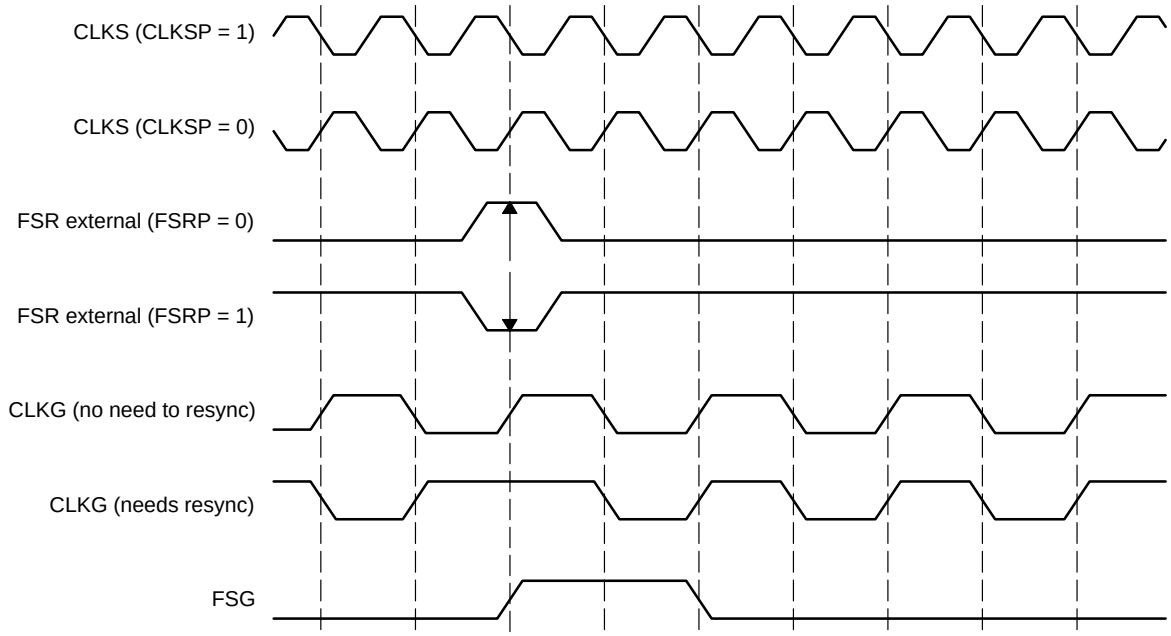
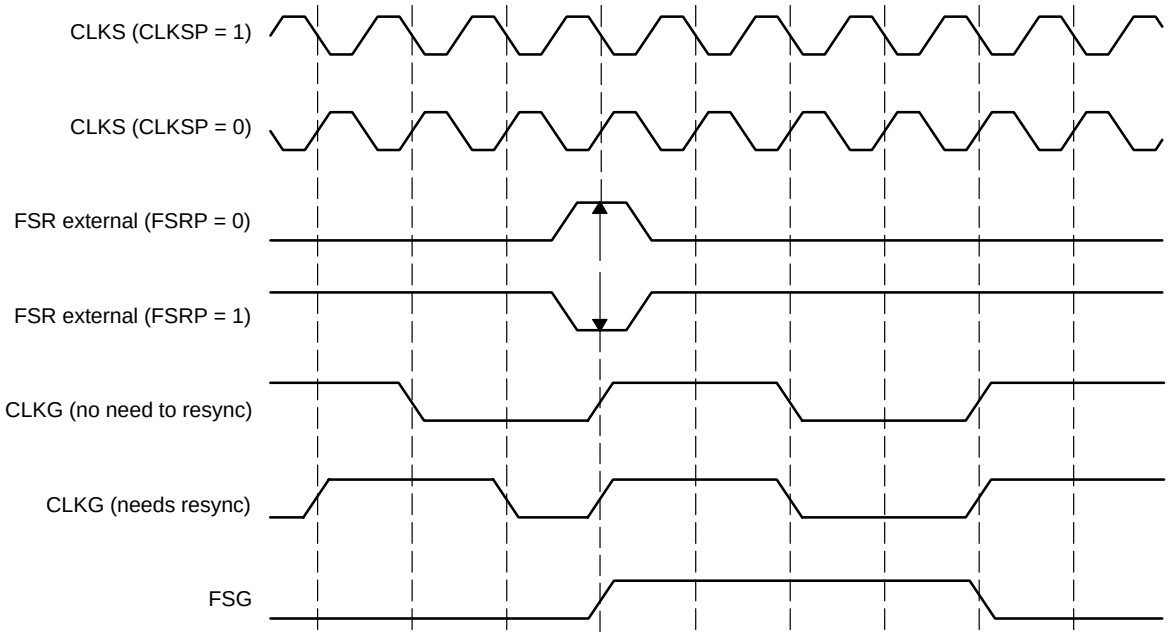


Figure 8–38 and Figure 8–39 shows this operation with various polarities of CLKS and FSR. These figures assume:

- FWID = 0, for a FSG one CLKG wide.

Figure 8–39. CLKG Synchronization and FSG generation when GSYNC = 1 and CLKGDV = 3



Note that FPER is not programmed since it is determined by the arrival of the next external frame sync pulse. The figure shows what happens to CLKG when it is initially in sync and GSYNC = 1 as well as when it is not in sync with the frame synchronization and GSYNC = 1.

When GSYNC = 1, the transmitter can operate synchronously with the receiver provided:

1) FSX is programmed to be driven by the sample rate generator frame sync FSG (FSGM = 1 in the SRGR and FSXM = 1 in the PCR). If the input FSR has the timing so that it can be sampled by the falling edge of CLKG, it can be used instead, by setting FSXM = 0 in the PCR and connecting FSR to FSX externally.

2) The sample rate generator clock should drive the transmit and receive bit clock (CLK(R/X)M = 1 in the SPCR). Therefore, the CLK(R/X) pin should not be driven by any other driving source.

8.5.2.5 Digital Loop Back Mode: DLB

Setting DLB = 1 in the SPCR enables digital loop back mode. During DLB mode, the DR, FSR, and CLKR are internally connected to DX, FSX, CLKX respectively, through multiplexers as shown in Figure 8–35. DLB mode allows testing of serial port code with a single DSP device. Figure 8–35 shows the multiplexing of receiver control inputs during digital loop back mode.

8.5.2.6 Receive Clock Selection: DLB, CLKRM

Table 8–15 shows how the digital loop back bit (DLB) and the CLKRM bit in the PCR can select the receiver clock. In digital loop back mode (DLB = 1), the transmitter clock drives the receiver. CLKRM determines whether the CLKR pin is an input or an output.

Table 8–15. Receive Clock Selection

DLB in SPCR	CLKRM in PCR	Source of Receive Clock	CLKR Pin
0	0	CLKR pin acts as an input driven by external clock and inverted as determined by CLKRP before being used.	Input
0	1	Sample Rate Generator Clock (CLKG) drives CLKR.	Output. CLKG inverted as determined by CLKRP before being driven out on CLKR.
1	0	CLKX_int drives the receive clock CLKR_int as selected and inverted as shown in Table 8–16.	High Impedance
1	1	CLKX_int drives CLKR_int as selected and inverted as shown in Table 8–16.	Output. CLKR (same as transmit) inverted as determined by CLKRP before being driven out.

8.5.2.7 Transmit Clock Selection: CLKXM

Table 8–16. Transmit Clock Selection

CLKXM in PCR	Source of Transmit Clock	CLKX Pin
0	External clock drives the CLKX input pin. CLKX is inverted as determined by CLKXP before being used.	Input
1	Sample rate generator clock, CLKG, drives transmit clock	Output. CLKG inverted as determined by CLKXP before being driven out on CLKX.

8.5.3 Frame Sync Signal Generation

Like data bit clocking, data frame synchronization is also independently programmable for the receiver and transmitter for all data delays. The FRST bit in the SPCR when set to 1 activates the frame generation logic to generate frame sync signals provided FSCM = 1 in SRGR. Frame sync programming options include:

- A frame pulse with programmable period between sync pulses, and programmable active width using the sample rate generator register (SRGR).
- The transmit portion may trigger its own frame sync signal generated by a DXR-to-XSR copy.
- Both the receive and transmit sections may independently select an external frame synchronization on the FSR and FSX pins, respectively.

Another method of generating frame sync pulses is to program FSGM = 0. This causes a frame sync to occur on every DXR to XSR copy. The data delays can be programmed as required. However, maximum packet frequency cannot be achieved in this method for data delays one and two. This limitation can be overcome by programming the frame ignore bit (R/X)FIG = 1.

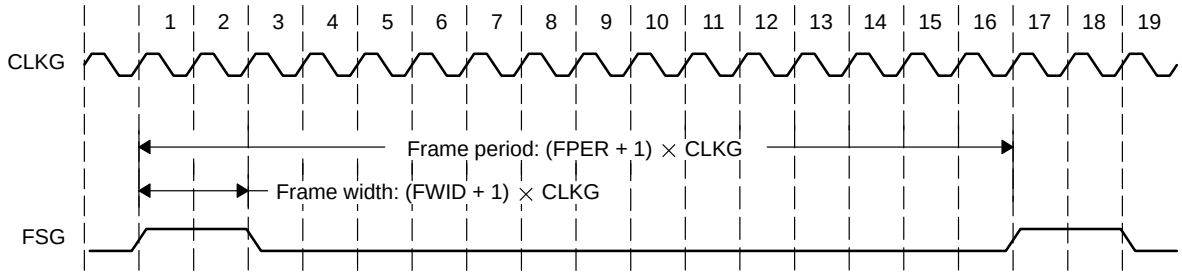
8.5.3.1 Frame Period and Frame Width: FPER and FWID

The FPER and FWID are implemented as down-counters. The FPER stage is a 12-bit down-counter which can count down the generated data clocks from 4095 to 0. The FWID stage in the sample rate generator is an 8-bit down counter. The FWID field controls the active width of the frame sync pulse. Both of these counters get loaded with their respective programmed value in FPER and FWID.

Once the sample rate generator is out of reset, FSG is in an inactive-low state. After this, when $\overline{\text{FRST}}$ = 1 and FSGM = 1 frame sync signals are generated. The frame width value (FWID+1) is counted down on every CLKG cycle until it reaches zero when FSG goes low. At the same time, the frame period value (FPER+1) is also counting down and when this value reaches zero, FSG goes high again indicating a new frame. We recommend that FWID be programmed to a value less than WDLEN(1/2).

Thus, the value of FPER+1 determines a frame length from 1 to 4096 data bits. When GSYNC = 1, FPER is a don't care value. Figure 8–40 shows a frame of period 16 CLKG periods (FPER = 15 or 00001111b). Thus, the value of FWID+1 determines a active frame pulse width ranging from 1 to 256 data bit clocks. Figure 8–40 shows a frame with an active width of 2 CLKG periods (FWID = 1).

Figure 8–40. Programmable Frame Period and Width



8.5.3.2 Receive Frame Sync Selection: DLB, FSRM, GSYNC

Table 8–17 shows how you may select various sources to provide the receive frame synchronization signal. Note that in digital loop back mode (DLB = 1) the transmit frame sync signal is used as the receive frame sync signal and that DR is connected to DX internally.

Table 8–17. Receive Frame Synchronization Selection

DLB in SPCR	FSR in PCR	GSYNC in SRGR	Source of Receive Frame Synchronization	FSR Pin
0	0	x	External frame sync signal drives the FSR input pin. This is then inverted as determined by FSRP before being used as FSR_int.	Input.
0	1	0	FSR_int driven by Sample Rate Generator Frame Sync signal (FSG), $\overline{\text{FRST}}$ = 1	Output. FSG inverted as determined by FSRP before being driven out on FSR pin.
0	1	1	FSR_int driven by Sample Rate Generator Frame Sync signal (FSG), $\overline{\text{FRST}}$ = 1	Input. The external frame sync input on FSR is used to synchronize CLKG and generate FSG.
1	0	0	FSX_int drives FSR_int. FSX is selected as shown in Table 8–18.	High Impedance.
1	X	1	FSX_int drives FSR_int and is selected as shown in Table 8–18.	Input. External FSR not used for frame synchronization but still used to synchronize CLKG and generate FSG since GSYNC = 1.
1	1	0	FSX_int drives FSR_int and is selected as shown in Table 8–18.	Output. Receive (same as transmit) frame synchronization inverted as determined by FSRP before being driven out.

8.5.3.3 Transmit Frame Sync Signal Selection: FSXM, FSGM

Table 8–18 shows how you can select the source of transmit frame synchronization pulses. The three choices are:

1) External frame sync input.
2) The sample rate generator frame sync signal, FSG.
3) A signal that indicates a DXR-to-XSR copy has been made.

Table 8–18. Transmit Frame Synchronization Selection

FSXM in PCR	FSGM in SRGR	Source of Transmit Frame Synchronization	FSX Pin
0	x	External frame sync input on FSX pin. This is inverted by FSXP before being used as FSX_int.	Input.
1	1	Sample Rate Generator Frame Sync signal (FSG) drives FSX_int. $\overline{\text{FRST}}$ = 1	Output. FSG inverted by FSXP before being driven out on FSX pin.
1	0	A DXR-to-XSR copy activates transmit frame sync signal.	Output. One bit clock wide signal inverted as determined by FSXP before being driven out on FSX pin.

8.5.3.4 Frame Detection for Initialization

To facilitate detection of frame synchronization, the receive and transmit CPU interrupts (RINT and XINT) may be programmed to detect frame synchronization by setting RINTM = XINTM = 10b, in the SPCR. Unlike other types of serial port interrupts, this mode can operate while the associated portion of the serial port is in reset (such as activating RINT when the receiver is in reset). In that case, the FS(R/X)M and FS(R/X)P still select the appropriate source and polarity of frame synchronization. Thus even when the serial port is in reset state, these signals are synchronized to CPU clock and then sent to the CPU in the form of RINT and XINT at the point at which they feed the receive and transmit portions of the serial port. Thus, a new frame synchronization pulse can be detected, after which the CPU can safely take the serial port out of reset.

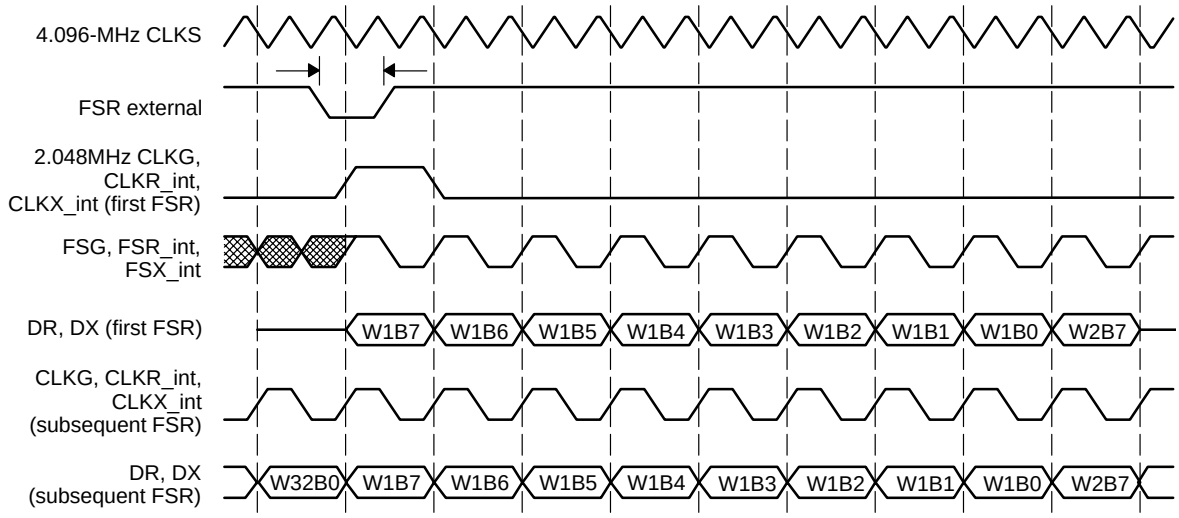
8.5.4 Examples

8.5.4.1 Double-Rate ST-BUS

Figure 8–41 shows McBSP configuration to be compatible with the Mitel ST-Bus. Note that this operation is running at maximum packet frequency.

- CLK(R/X)M = 1, CLK(R/X)_int generated internally by sample rate generator
- GSYNC = 1, synchronize CLKG with external frame sync signal input on FSR. Note that CLKG is not synchronized (runs freely) until frame sync signal is active. Also note that FSR is regenerated internally to form a minimum pulse width.
- CLKSM = 1, external clock (CLKS) drives the sample rate generator
- CLKSP = 1, falling edge of CLKS generates CLKG and thus CLK(R/X)_int
- CLKGDV = 1, receive clock (shown as CLKR) is half of CLKS frequency
- FS(R/X)P = 1, active-low frame sync pulse
- (R/X)FRLEN1 = 11111b, 32 words per frame
- (R/X)WDLEN1 = 0, 8-bit word
- (R/X)PHASE = 0, single phase frame and thus (R/X)FRLEN2 = (R/X)WDLEN2 = X
- (R/X)DATDLY = 0, no data delay

Figure 8–41. ST-BUS and MVIP Example

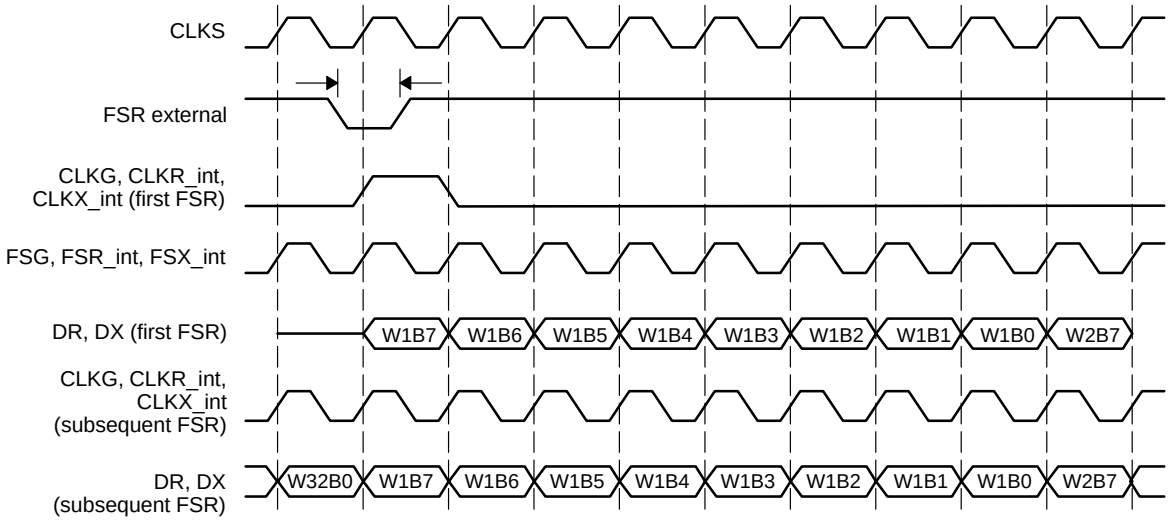


8.5.4.2 Single Rate ST-BUS Clock

This example is the same as the ST-BUS example except for the following:

- ❑ CLKGDV = 0, CLKS drives CLK(R/X)_int without any divide down (single rate clock).
- ❑ CLKSP = 0, rising edge of CLKS generates internal clocks CLKG, CLK(R/X)_int.

Figure 8–42. Single Rate Clock Example



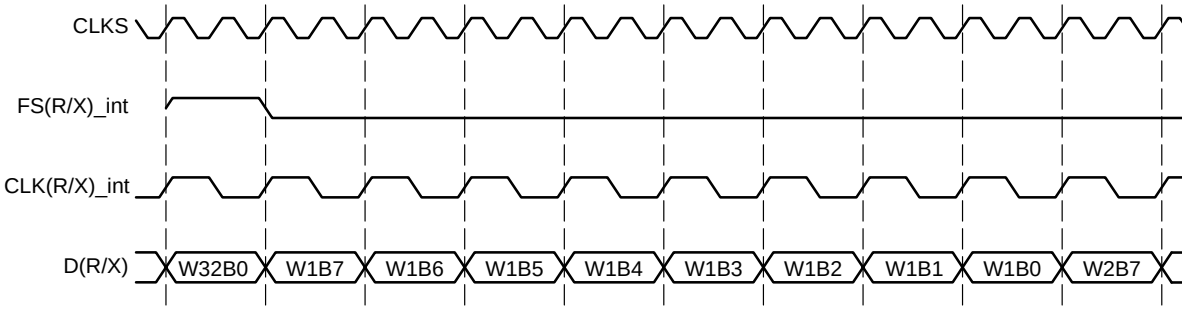
The rising edge of CLKS is used to detect the external FSR. This external frame sync pulse is used to re-synchronize internal McBSP clocks and generate frame sync for internal use. Note that the internal frame sync is generated so that it is wide enough to be detected on the falling edge of internal clocks.

8.5.4.3 Double Rate Clock Example

This example is the same as the ST-BUS example except for the following:

- CLKSP = 0, rising edge of CLKS generates CLKG and thus CLK(R/X)
- CLKGDV = 1, CLKG and thus CLKR_int and CLKX_int frequency is half of CLKS.
- GSYNC = 0, CLKS drives CLKG. CLKG runs freely and is not re-synchronized by FSR.
- FS(R/X)M = 0, frame synchronization is externally generated. The framing pulse is wide enough to be detected.
- FS(R/X)P = 0, active-high input frame sync signal
- (R/X)DATDLY = 1, data delay of 1-bit

Figure 8–43. Double Rate Clock Example



8.6 Multichannel Selection Operation

Multiple channels can be independently selected for the transmitter and receiver by configuring the McBSP with a single phase frame (see subsection 8.3.4.3). The number of words per frame represented by (R/X)FRLEN1, denotes the number of channels available for selection.

Each frame represents a time-division multiplexed (TDM) data stream. In using time-division multiplexed data streams, the CPU may only need to process a few of them. Thus, to save memory and bus bandwidth, multichannel selection allows independent enabling of particular channels for transmission and reception. Up to 32 channels in an up to 128 channel bit stream can be enabled.

If a receive channel is not enabled:

- RRDY is not set to 1 upon reception of the last bit of the word.
- RBR is not copied to DRR upon reception of the last bit of the word. Thus, RRDY is not set active. This feature also implies that no interrupts or synchronization events are generated for this word.

If a transmit channel is not enabled:

- DX is in high impedance.
- A DXR-to-XSR transfer is not automatically triggered at the end of serial transmission of the related word.
- $\overline{\text{XEMPTY}}$ and XRDY similarly are not affected by the end of transmission of the related serial word .

A transmit channel which is enabled can have its data masked or transmitted. When masked, the DX pin will be forced to high impedance although the transmit channel is enabled.

8.6.1 Multichannel Operation Control Registers

The following control registers are used in multichannel operation:

1) The Multichannel Control (MCR) Register
2) The Transmit Channel Enable (XCER) Register
3) The Receive Channel Enable (RCER) Register

The use of these registers in controlling multichannel operation is described in the following sections.

Figure 8–44. Multichannel Control Register

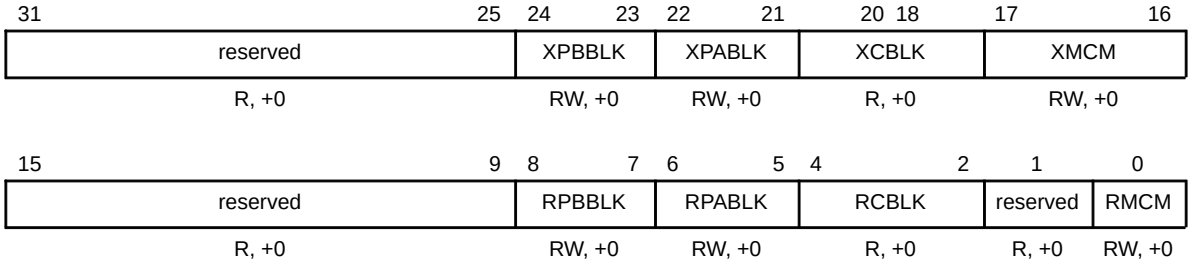


Table 8–19. Multichannel Control Register Bit-Field Descriptions

Name	Function	Section
RMCM	Receive Multichannel Selection Enable RMCM = 0, all 128 channels enabled. RMCM = 1, all channels disabled by default. Required channels are selected by enabling RP(A/B)BLK and RCER appropriately.	8.6.2
XMCM	Transmit Multichannel Selection Enable XMCM = 00b, all channels enabled without masking (DX is always driven during transmission of data).† XMCM = 01b, all channels disabled and therefore masked by default. Required channels are selected by enabling XP(A/B)BLK and XCER appropriately. Also, these selected channels are not masked and therefore DX is always driven. XMCM = 10b, all channels enabled, but masked. Selected channels enabled via XP(A/B)BLK and XCER are unmasked. XMCM = 11b, all channels disabled and therefore masked by default. Required channels are selected by enabling RP(A/B)BLK and RCER appropriately. Selected channels can be unmasked by RP(A/B)BLK and XCER. This mode is used for symmetric transmit and receive operation.	8.6.3
RCBLK/ XCBLK	Receive/Transmit Current Block. (R/X)CBLK = 000b, Block 0. Channel 0 to channel 15 (R/X)CBLK = 001b, Block 1. Channel 16 to channel 31 (R/X)CBLK = 010b, Block 2. Channel 32 to channel 47 (R/X)CBLK = 011b, Block 3. Channel 48 to channel 63 (R/X)CBLK = 100b, Block 4. Channel 64 to channel 79 (R/X)CBLK = 101b, Block 5. Channel 80 to channel 95 (R/X)CBLK = 110b, Block 6. Channel 96 to channel 111 (R/X)CBLK = 111b, Block 7. Channel 112 to channel 127	8.6.3.2

Table 8–19. Multichannel Control Register Bit-Field Descriptions (Continued)

Name	Function	Section
RPBBLK/ XPBBLK	Receive/Transmit Partition B Block. (R/X)PBBLK = 00b, Block 1. Channel 16 to channel 31 (R/X)PBBLK = 01b, Block 3. Channel 48 to channel 63 (R/X)PBBLK = 10b, Block 5. Channel 80 to channel 95 (R/X)PBBLK = 11b, Block 7. Channel 112 to channel 127	8.6.3
RPABLK/ XPABLK	Receive/Transmit Partition A Block. (R/X)PABLK = 00b, Block 0. Channel 0 to channel 15 (R/X)PABLK = 01b, Block 2. Channel 32 to channel 47 (R/X)PABLK = 10b, Block 4. Channel 64 to channel 79 (R/X)PABLK = 11b, Block 6. Channel 96 to channel 111	8.6.3

† DX is masked or driven to hi–Z during (a) inter–packet intervals, (b) when a channel is masked regardless of whether it is enabled, or (c) when a channel is disabled.

8.6.2 Enabling Multichannel Selection

Multichannel mode can be enabled independently for reception and transmission by setting RMCM = 1 and XMCM to a non-zero value in the MCR, respectively.

8.6.3 Enabling and Masking of Channels

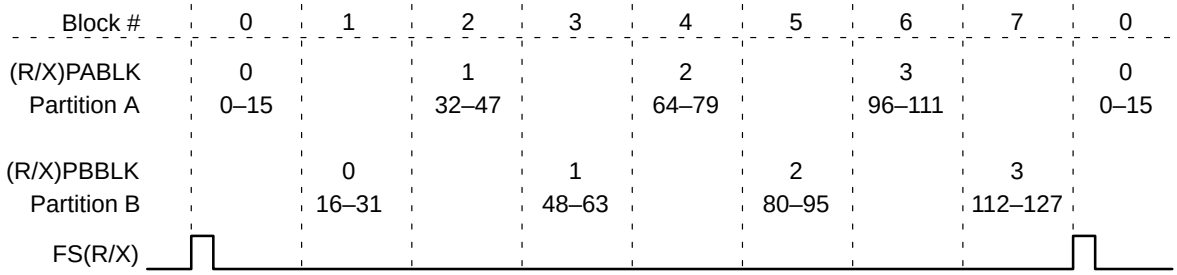
A total of 32 out of the available 128 channels may be enabled at any given point in time. The 128 channels comprise eight blocks (0 through 7) and each block has 16 contiguous channels. Further, even-numbered blocks 0, 2, 4, and 6 belong to Partition A, and odd-numbered blocks 1, 3, 5, and 7 belong to Partition B.

The number of channels enabled can be updated during the course of a frame to allow any arbitrary group of channels to be enabled. This feature is accomplished using an alternating ping-pong scheme controlling two blocks (one odd-numbered and other even-numbered) of 16 contiguous channels each, at any given time within the frame. Thus one block belongs to Partition A and the other to Partition B.

Any two out of the eight 16-channel blocks may be selected, yielding a total of 32 channels that can be enabled. The blocks are allocated on 16-channel boundaries within the frame as shown in Figure 8–45. (R/X)PABLK and

(R/X)PBBLK fields in the MCR determine the blocks that get selected in partition A and B respectively. This enabling is performed independently for transmit and receive.

Figure 8–45. Channel Enabling by Blocks in Partition A and B



Transmit data masking allows a channel enabled for transmit to have its DX pin set to high impedance state during its transmit period. In systems where symmetric transmit and receive provides software benefits, this feature allows transmit channels to be disabled on a shared serial bus. A similar feature is not needed for receive as multiple reception cannot cause serial bus contention.

Note:

DX is masked or driven to hi-Z during (a) inter-packet intervals, (b) when a channel is masked regardless of whether it is enabled, or (c) when a channel is disabled.

The following gives a description of the multichannel operation during transmit for various XMCM values:

- XMCM = 00b: The serial port will transmit data over the DX pin for as many number of words as programmed in XFRLEN1. Thus, DX is driven during transmit.
- XMCM = 01b: Required channels or only those words that need to be transmitted are selected via XP(A/B)BLK and XCER. Therefore only these selected words will be written to DXR and ultimately transmitted. In other words, if XINTM = 00b which implies that an XINT will be generated for every DXR-to-XSR copy, the number of XINT generated will be equal to the number of channels selected via XCER (and NOT equal to XFRLEN1).
- XMCM = 10b: For this case, all channels are enabled which means all the words in a data frame (XFRLEN1) will be written to DXR and DXR-to-XSR

copy occurs at their respective times. However, DX will be driven only for those channels that are selected via XP(A/B)BLK and XCER and tri-stated otherwise. In this case, if XINTM = 00b, the number of interrupts generated due to every DXR-to-XSR copy would equal the number of words in that frame (XFRLEN1).

- XMCM = 11b: This mode is basically a combination of XMCM = 01b and 10b cases so that symmetric transmit and receive operation is achieved. All channels are disabled and therefore DR and DX are in a state of high impedance. For receiving, a RBR-to-DRR copy occurs only for those channels that are selected via RP(A/B)BLK and RCER. If RINT were to be generated for every RBR-to-DRR copy, it would occur as many times as the number of channels selected in RCER (and NOT the number of words programmed in RFRLEN1). For transmitting, the same block that is used for reception is used in order to maintain symmetry and thus XP(A/B)BLK is a don't care. DXR is loaded and DXR-to-XSR copy occurs for all the channels that are enabled via RP(A/B)BLK . However, DX will be driven only for those channels that are selected via XCER. Note that the channels enabled in XCER can only be a subset or same as those selected in RCER. Therefore, if XINTM = 00b, transmit interrupts to the CPU would be generated as many times as the number of channels selected in RCER (not XCER).

Figure shows the activity on the McBSP pins for all the above modes with the following conditions:

- (R/X)PHASE = 0, single phase frame for multichannel selection enabled
- FRLEN1 = 011b, 4-word frame
- WDLEN1 = any valid serial word length

Please note that in the following illustrations, the arrows showing where the various events occur are only a sample indication. Wherever possible, there is a time window in which these events could occur.

Figure 8–46. XMCM Operation

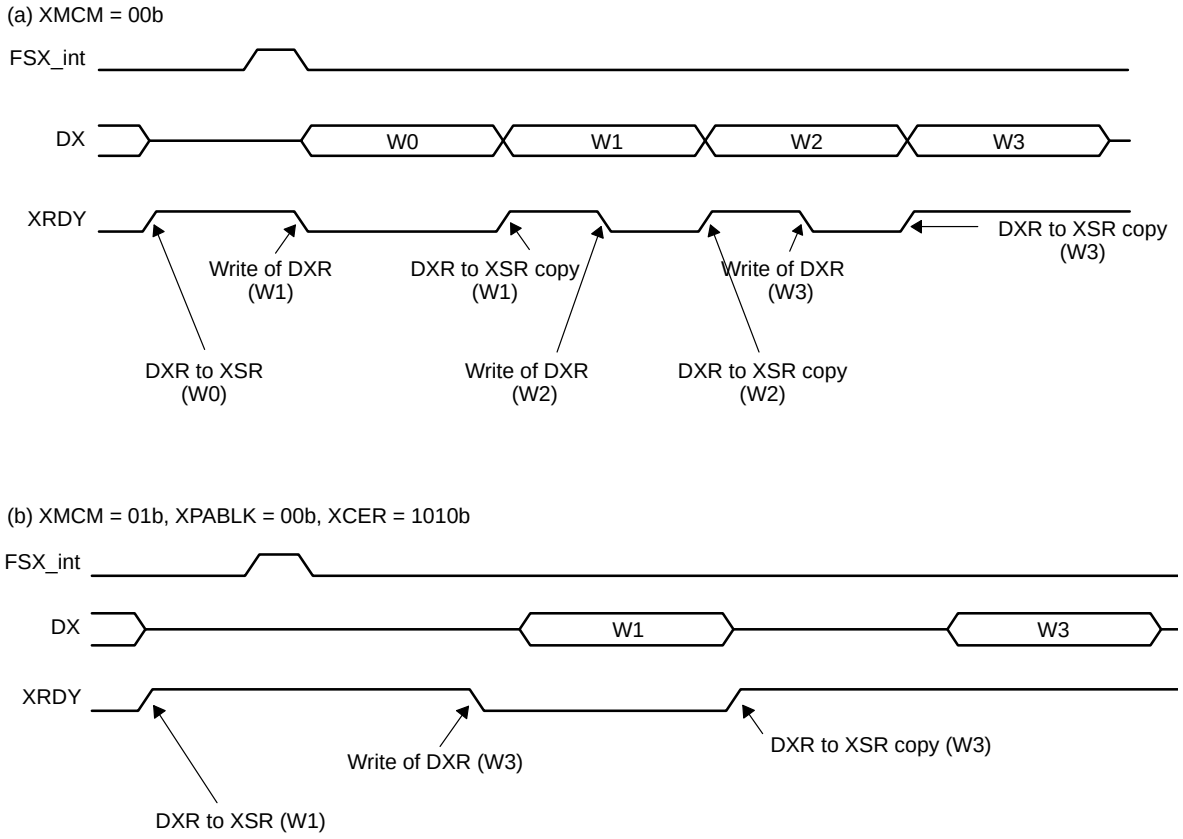
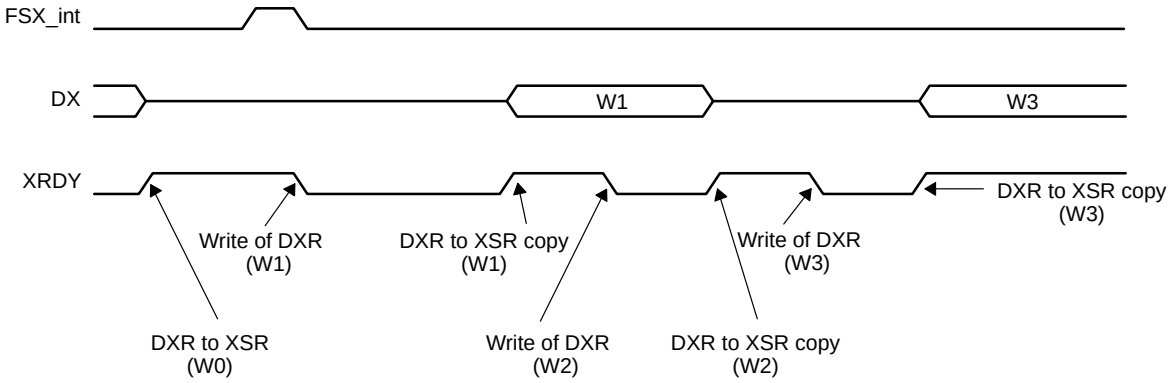
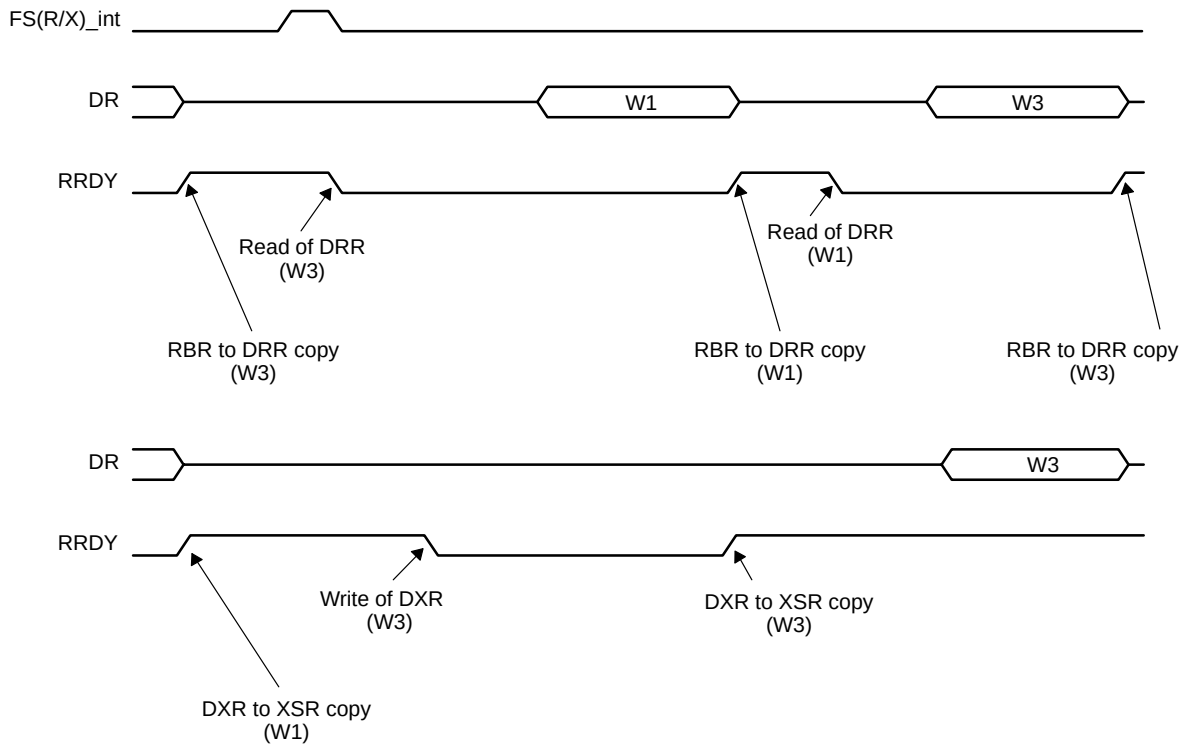


Figure 8–46. XMCM Operation (Continued)

(c) XMCM = 10b, XPABLK = 00b, XCER = 1010b



(d) XMCM = 11b, RPABLK = 00b, XPABLK = X, RCER = 1010b, XCER = 1000b



8.6.3.1 Channel Enable Registers: (R/X)CER

The Receive Channel Enable (RCER) and Transmit Channel Enable (XCER) registers are used to enable any of the 32 channels for receive and transmit, respectively. Out of the 32 channels, 16 channels belong to a block in partition A and the other 16 belong to a block in partition B. They are shown in Figure 8–47 and Figure 8–48. (R/X)CEA and (R/X)CEB register fields shown in Table 8–20 enable channels within the 16 channel wide blocks in partitions A and B, respectively. The (R/X)PABLK and (R/X)PBBLK fields in the MCR select which 16-channel blocks get selected.

Figure 8–47. Receive Channel Enable Register (RCER)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RCEB 15	RCEB 14	RCEB 13	RCEB 12	RCEB 11	RCEB 10	RCEB 9	RCEB 8	RCEB 7	RCEB 6	RCEB 5	RCEB 4	RCEB 3	RCEB 2	RCEB 1	RCEB 0
RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RCEA 15	RCEA 14	RCEA 13	RCEA 12	RCEA 11	RCEA 10	RCEA 9	RCEA 8	RCEA 7	RCEA 6	RCEA 5	RCEA 4	RCEA 3	RCEA 2	RCEA 1	RCEA 0
RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0

Figure 8–48. Transmit Channel Enable Register (XCER) Diagram

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
XCEB 15	XCEB 14	XCEB 13	XCEB 12	XCEB 11	XCEB 10	XCEB 9	XCEB 8	XCEB 7	XCEB 6	XCEB 5	XCEB 4	XCEB 3	XCEB 2	XCEB 1	XCEB 0
RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XCEA 15	XCEA 14	XCEA 13	XCEA 12	XCEA 11	XCEA 10	XCEA 9	XCEA 8	XCEA 7	XCEA 6	XCEA 5	XCEA 4	XCEA 3	XCEA 2	XCEA 1	XCEA 0
RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0

Table 8–20. Receive/Transmit Channel Enable Register Bit-Field Description

Name	Function
(R/X)CEA*n* 0 ≤ n ≤ 15	Receive/Transmit Channel Enable (R/X)CEA *n* = 0, Disables reception/transmission of *n*th channel in an even-numbered block in partition A (R/X)CEA *n* = 1, Enables reception/transmission of *n*th channel in an even-numbered block in partition A
(R/X)CEB*n* 0 ≤ n ≤ 15	Receive/Transmit Channel Enable (R/X)CEB *n* = 0, Disables reception/transmission of *n*th channel in odd-numbered block in partition B. (R/X)CEB *n* = 1, Enables reception/transmission of *n*th channel in odd-numbered block in partition B.

8.6.3.2 Changing Channel Selection

Using the multichannel selection feature, a static group of 32 channels may be enabled, and will remain enabled with no CPU intervention until this allocation requires modification. An arbitrary number, group, or all of the words/channels within a frame can be accessed, by updating the block allocation registers during the course of the frame in response to the end-of-block interrupts (see 8.6.3.3 for Update Interrupts).

However, the user must be careful when changing the selection, not to affect the currently selected block. The currently selected block is readable through the RCBLK and XCBLK fields in the MCR for receive and transmit respectively. The associated channel enable register cannot be modified if it is selected by the appropriate (R/X)P(A/B)BLK register to point toward the current block. Similarly the (R/X)PABLK and (R/X)PBBLK fields in the MCR cannot be modified while pointing to or being changed to point to the currently selected block. Note that if the total number of channels is 16 or less, the current partition is always pointed to. In this case, only a reset of the serial port can change enabling.

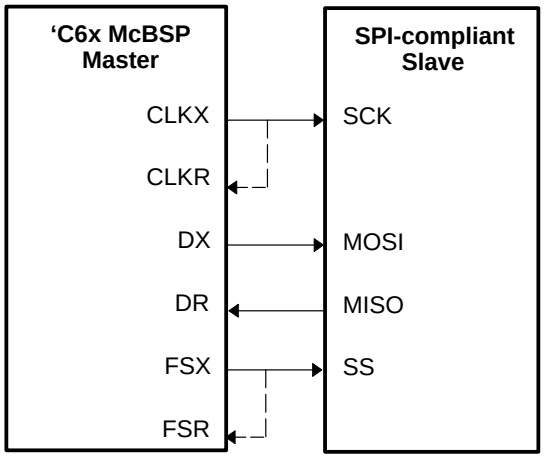
8.6.3.3 End-of-Block or End-of-Frame Interrupt

At the end of every 16-channel block boundary during multichannel operation, the receive interrupt (RINT) or transmit interrupt (XINT) to the CPU is generated if RINTM = 01b or XINTM = 01b in the SPCR, respectively. This interrupt indicates a new partition has been crossed. You can then check the current partition and change the selection of blocks in the A and/or B partitions if they do not point to the current block. These interrupts are two CPU clock long active high pulses. Note that if RINTM = XINTM = 01b when (R/X)MCM = 0 (non-multichannel operation), it does not generate any interrupts.

8.7 SPI™ Protocol: CLKSTP

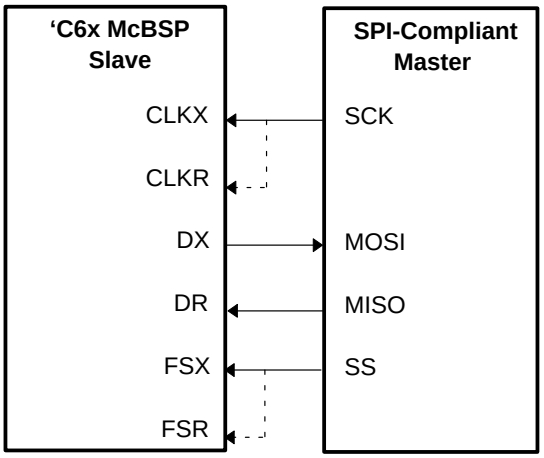
A system conforming to this protocol is a master-slave configuration. The SPI™ protocol is a four wire interface comprising of serial data in (Master In Slave Out or MISO), serial data out (Master Out Slave In or MOSI), shift clock (SCK), and an active low slave enable ($\overline{SS}$) signal. Communication between the master and slave is determined by the presence or absence of the master clock. Therefore, in absence of a dedicated frame synchronization signal, the data transfer is initiated by the detection of the master clock and is terminated on absence of the master clock. The slave has to be enabled during this period of transfer. In the case of McBSP being the master, the slave enable is derived from the master transmit frame sync pulse, FSX. An example block diagram of McBSP as a master and as a slave is shown in Figure 8–49 and Figure 8–50.

Figure 8–49. SPI Configuration: McBSP as the Master



The clock stop mode (CLKSTP) in the McBSP provides compatibility with the SPI™ protocol. Clock stop mode works with only single-phase frames ((R/X)PHASE = 0) and one word per frame. The word sizes supported by the McBSP are programmable for 8-, 12-, 16-, 20-, 24-, or 32-bit operation. Clock stop mode bit-field (CLKSTP) in the SPCR in conjunction with CLKXP bit in the PCR, allows serial clocks to be stopped between transfers using one of four possible timing variations as shown in Table 8–21. When the McBSP is configured to operate in SPI mode, both the transmitter and the receiver operate together as a master or a slave. The McBSP is a master when it generates clocks. The master's transmit clock drives its own receive clock and the clocks to the slave device. In conjunction with CLKSTP enabled, CLKXM = 1 (in PCR) indicates that the McBSP is a master, and CLKXM = 0 indicates that the McBSP is an SPI slave.

Figure 8–50. SPI Configuration: McBSP as the Slave



The slave enable signal serves to enable the serial data input and output driver on the slave device (device not outputting clock). Some devices do not have the slave enable pin. In order to interface to devices with and without the slave enable, an alternative framing scheme is used.

When the McBSP is the SPI master device, CLKX should be configured as an output, and FSX should be configured as as an output (generated with a load of DXR) which is connected to its own FSR (as an input) and to the slave enable ($\overline{SS}$) input on the slave device, if required. The FSGM bit in SRGR is zero so that DXR to XSR transfer generates the FSX and hence FSR and slave enable signal. The slave should be enabled before beginning the transfer which means that the XDATDLY = 1 should be programmed. The CLKGDV (clock divide ratio) in SRGR should be programmed to generate the required SPI data rate. The McBSP generates a continuous clock (CLKX) internally and gates the clock off (stops the clock) to the external interface when transfers are complete. In this case receive clock is provided from the internal, continuously running version, so the receiver and transmitter both work internally as if clocks do not stop.

When the McBSP is a slave device, the internal serial port logic performs transfers using only the exact number of input clock pulses per data bits. CLK(R/X) pins are configured as inputs and are driven by the SPI master clock in the system. If the master device provides a slave enable signal, it is connected to FSX/FSR and is used in its asynchronous form. Thus, transmit and receive frame sync mode bits, FSXM and FSRM should be set to 0. FSX, then, controls only the initial drive of data to the DX pin. The (R/X)DATDLY in this case (McBSP slave) should be zero so that data is driven out or shifted in on the same edge where the frame sync occurs. If the master device does not provide a slave enable, the external FSR/FSX pins are connected to an active level (as defined

by FSX/FSR polarity control bits), and data is driven to the DX pin as soon as DXR is loaded. The input clock and frame sync from the master is synchronized to the CPU clock to allow reset. It is required that the CLKGDV be a value such that the CLKG is at least 8 times the SPI data rate. The first data to be transmitted is available on the DX pin, but is enabled only after detection of SPI clock.

In clock stop mode, for both the transmitter and receiver, stopping of clocks is handled differently depending on whether the clocks are externally or internally generated. In the case where clocks are externally generated, the external device that is generating clocks holds clocks appropriately between transmitted words. When the serial port clock is internally generated, this clock runs continuously, and is simply gated off (clock stops) to the external pin when the transfer is complete. In this case, transfers are performed in the same fashion as with external free running clocks.

Table 8–21. SPI-Mode Clock Stop Scheme

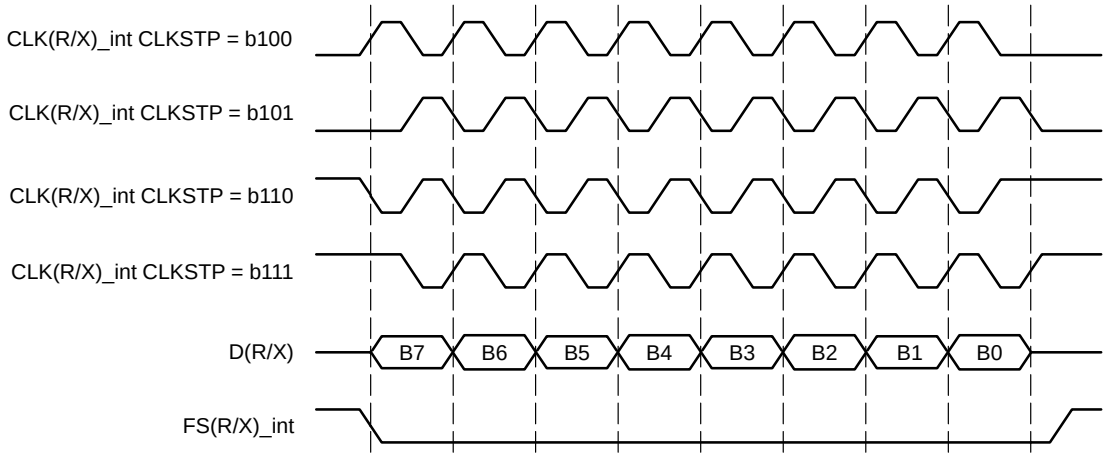
CLKSTP	CLKXP	Clock Scheme
0X	X	Clock Stop Mode Disabled. Clock enabled for non-SPI mode.
10	0	Low inactive state without delay: The McBSP transmits data on the rising edge of CLKX and receives data on the falling edge of CLKR.
11	0	Low inactive state with delay: The McBSP transmits data one half cycle ahead of the rising edge of CLKX and receives data on the rising edge of CLKR.
10	1	High inactive state without delay: The McBSP transmits data on the falling edge of CLKX and receives data on the rising edge of CLKR.
11	1	High inactive state with delay: The McBSP transmits data one half cycle ahead of the falling edge of CLKX and receives data on the falling edge of CLKR.

CLKSTP and CLKXP bits selects the appropriate clock scheme for a particular SPI interface as shown in Table 8–21 and Figure 8–51. CLKSTP bit-field in the SPCR selects one of the following:

1) Whether clock stop mode is enabled or not.
2) In clock stop mode, whether the clock is high or low when stopped.
3) In clock stop mode, whether first clock edge occurs at the start of the first data bit or at the middle of the first data bit.

The CLKXP bit selects the edge on which data is transmitted (driven) and received (sampled) as shown in Table 8–21.

Figure 8–51. Clock Stop Mode Options



8.7.1 McBSP Initialization for SPI mode

The operation of the serial port during device reset, transmitter reset, and receiver reset is described in subsection 8.3.1. For the case when the McBSP needs to be configured for SPI receive/transmit operation as a master or a slave, the following steps have to be followed for proper initialization:

1) Set $\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = 0 in SPCR.

2) Program the necessary McBSP configuration registers (and not the data registers) listed in Table 8–2 as required when the serial port is in reset state ($\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = 0) *except* for CLKSTP bits which should be 0Xb. Note that CLKXP is also programmed in this stage.

3) Wait two bit clocks for the McBSP to re-initialize.

4) Write the desired value into the CLKSTP bit-fields in the SPCR.

5) Set $\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = 1 to enable the serial port. Note that the value written to the SPCR at this time should have only the reset bits changed to 1 and the remaining bit-fields should have the same value as in Step 2 and 4 above.

6) Wait two bit clocks for the receiver and transmitter to become active.

8.8 McBSP Pins as General Purpose I/O

Two conditions allow the serial port pins (CLKX, FSX, DX, CLKR, FSR, and DR) to be used as general purpose I/O rather than serial port pins:

1) The related portion (transmitter or receiver) of the serial port is in reset; ($\overline{\text{R/X}}$)$\overline{\text{RST}}$ = 0 in the SPCR, and

2) General purpose I/O is enabled for the related portion of the serial port; (R/X)IOEN = 1 in the PCR.

Figure 8–3 has bits that configure each of the McBSP pins as general purpose inputs or outputs. Table 8–22 shows how this is achieved. In the case of FS(R/X), FS(R/X)M = 0 configures the pin as an input and FS(R/X)M = 1 configures that pin as an output. When configured as an output, the value driven on FS(R/X) is the value stored in FS(R/X)P. If configured as an input, the FS(R/X)P becomes a read only bit that reflects the status of that signal. CLK(R/X)M and CLK(R/X)P work similarly for CLK(R/X). When the transmitter is selected as general purpose I/O, the value of the DX_STAT bit in the PCR is driven onto DX. DR is always an input and its value is held in the DR_STAT bit in the PCR. To configure CLKS as a general purpose input, both the transmitter and receiver have to be in reset state and (R/X)IOEN = 1, because it is always an input to the McBSP and affects both transmit and receive operations.

Table 8–22. Configuration of Pins as General Purpose I/O

Pin	General Purpose I/O Enabled by Setting Both	Selected as Output	Output Value Driven From	Selected as Input	Input Value Readable on
CLKX	$\overline{\text{XRST}}$ = 0 XIOEN = 1	CLKXM = 1	CLKXP	CLKXM = 0	CLKXP
FSX	$\overline{\text{XRST}}$ = 0 XIOEN = 1	FSXM = 1	FSXP	FSXM = 0	FSXP
DX	$\overline{\text{XRST}}$ = 0 XIOEN = 1	always	DX_STAT	never	does not apply
CLKR	$\overline{\text{RRST}}$ = 0 RIOEN = 1	CLKRM = 1	CLKRP	CLKRM = 0	CLKRP
FSR	$\overline{\text{RRST}}$ = 0 RIOEN = 1	FSRM = 1	FSRP	FSRM = 0	FSRP
DR	$\overline{\text{RRST}}$ = 0 RIOEN = 1	never	does not apply	always	DR_STAT
CLKS	$\overline{\text{RRST}}$ = $\overline{\text{XRST}}$ = 0 RIOEN = XIOEN = 1	never	does not apply	always	CLKS_STAT

Chapter 9

Timer

This chapter describes the 32-bit timer functionality, registers and signals for the TMS320C62xx.

Topic **Page**

9.1 Overview

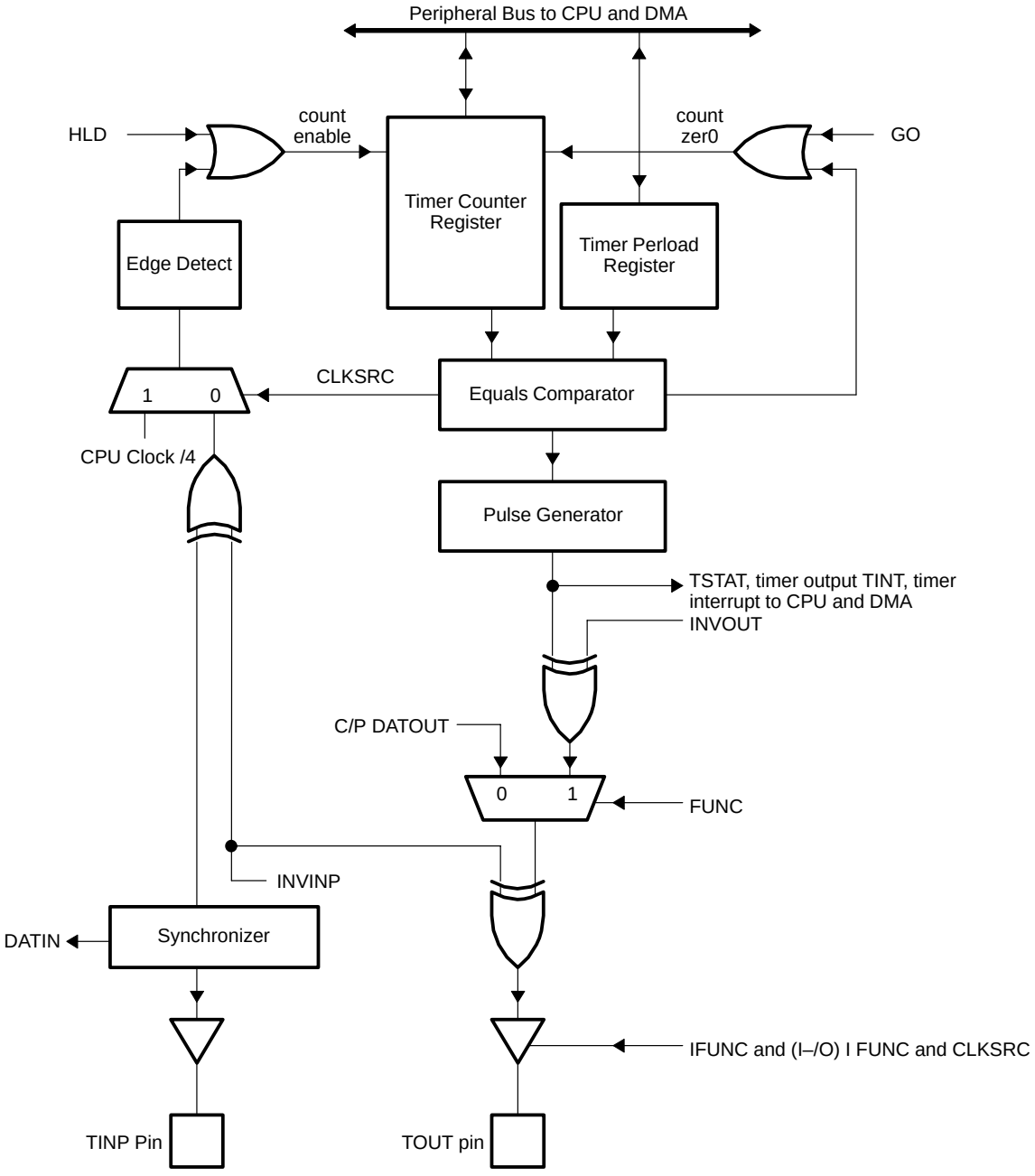
The device has two 32-bit general-purpose timers that you can use to:

- Time events
- Count events
- Generate pulses
- Interrupt the CPU
- Send synchronization events to the DMA

The timer has two signaling modes and can be clocked by an internal or an external source. The timer has an input pin and an output pin. The input and output pins, (TINP and TOUT) can function as timer clock input and clock output. They can also be configured for general-purpose input and output, respectively.

With an internal clock, for example, the timer can signal an external A/D converter to start a conversion, or it can trigger the DMA controller to begin a data transfer. With an external clock, for example, the timer can count external events and interrupt the CPU after a specified number of events. Figure 9–1 shows a block diagram of the timer.

Figure 9–1. Timer Block Diagram



9.2 Timer Registers

Table 9–1 describes the three registers that configure timer operation.

Table 9–1. Timer Registers

Hex Byte Address				
Timer 0	**Timer 1**	**Name**	**Description**	**Section**
01940004	01980004	Timer Period	Contains the number of timer input clock cycles to count. This number controls the TSTAT signal frequency.	9.2.1
01940000	01980000	Timer Control	Determines the operating mode of the timer, monitors the timer status, and controls the function of the TIM pin.	9.2.2
01940008	01980008	Timer Counter	Current value of the incrementing counter.	9.2.3

9.2.1 Timer Control Register

Figure 9–2 shows the timer control register. Table 9–2 describes the bitfields in this register.

Figure 9–2. Timer Control Register Diagram

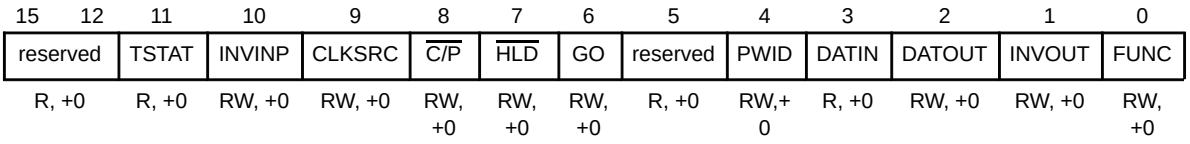


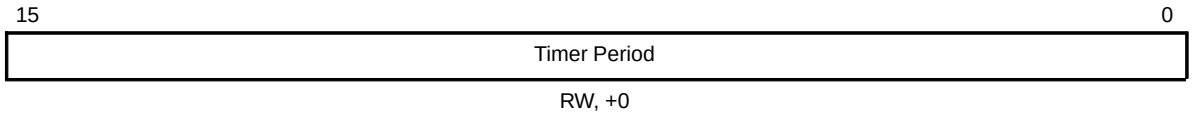
Table 9–2. Timer Control Register Field Description

Bitfield	Description	Section
FUNC	Function of TIM pin. FUNC = 0, TOUT is a general-purpose output pin. FUNC = 1, TOUT is a timer pin.	9.5
DATOUT	Data output. When FUNC = 0 Data Out, Value driven out on TOUT. When FUNC = 1, TSTAT driven on TOUT after inversion by INVOUT.	
DATIN	Data in. Value on TINP pin.	
GO	GO bit. Resets and starts the timer counter. GO = 0, no effect to timer. GO = 1, and HLD = 0, counter register zeroed and begins counting on next clock.	9.3
$\overline{\text{HLD}}$	Hold. Counter may be read or written regardless of HLD value. $\overline{\text{HLD}}$ = 0, counter disabled and held in current state. $\overline{\text{HLD}}$ = 1, counter allowed to count.	9.3
$\overline{\text{C/P}}$	Clock/pulse mode. $\overline{\text{C/P}}$ = 0, pulse mode, TSTAT active one CPU clock after timer reaches timer period. PWID determines when it goes inactive. $\overline{\text{C/P}}$ = 1, clock mode, TSTAT has 50% duty cycle with high and low periods each one countdown period wide.	9.6
PWID	Pulse Width. Only used in pulse mode ($\overline{\text{C/P}}$ = 0). The TSTAT goes inactive one clock after the timer counter does not equal PWID (0 or 1 as programmed.)	9.6
CLKSRC	Timer Input Clock Source CLKSRC = 0, value on TINP pin. CLKSRC = 1, CPU clock/4.	9.5
INVINP	TINP Inverter Control. Only affects operation if CLKSEC = 0. INVINP = 0, uninverted TINP drives timer. INVINP = 1, inverted TINP drives timer.	9.5
TSTAT	Timer status. Value of timer output.	9.6
INVOUT	TOUT Inverter Control. Only affects operator if FUNC = 1. INVINP = 0, uninverted TSTAT drives TOUT. INVINP = 1, inverted TSTAT drives TOUT.	

9.2.2 Timer Period Register

The timer period register (Figure 9–3) contains the number of timer input clock cycles to count. This number controls the frequency of TSTAT.

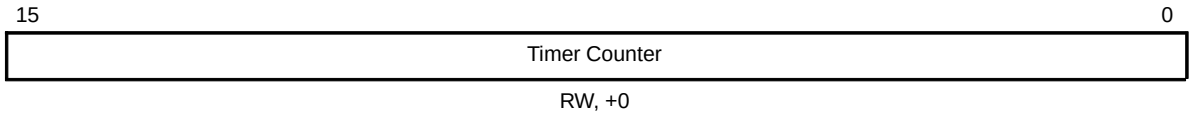
Figure 9–3. Timer Period Register Diagram



9.2.3 Timer Counter Register

The timer counter register (Figure 9–4) increments whenever enabled to count. Upon reaching the value in the timer period register, it resets to 0 on the next CPU clock. each cycle of the timer input clock.

Figure 9–4. Timer Counter Register Diagram



9.3 Resetting the Timer and Enabling Counting: GO and $\overline{HLD}$

Table 9–3 shows how the GO and HLD- enable basic features of timer operation.

Table 9–3. Timer GO and HLD Field Operation

Operation	GO	$\overline{HLD}$	Description
Holding the timer	0	0	Counting is disabled.
Restarting the timer after hold	0	1	Timer proceeds from state before hold. The timer counter is NOT reset.
Reserved	1	0	undefined
Starting the timer	1	1	Timer counter resets to 0 and starts counting whenever enabled. Once set, GO self-clears.

9.3.1 Initializing the Timer

Configuring a timer requires three basic steps:

1) If the timer is not currently held, place the timer in hold. Note that after device reset, the timer is already in the hold state.

2) Write the desired value to the timer period register.

3) Start the timer by setting the GO and HLD bits of the timer control register to 1 and simultaneously writing the desired values to the timer control register.

9.4 Timer Counting

The timer counter runs at the CPU clock rate. However, counting is enabled by on the low-to-high transition from one CPU clock to the next of one of the timer count enable source. This transition is detected by the edge detect circuit shown in Figure 9–1. Each time an active transition is detected a one CPU clock wide clock enable pulse is generated. To the user, this makes the counter appear as if it were getting clocked by the count enable source. Thus, this count enable source is referred to as the clock source.

Upon reaching a value equal to the timer period register, the timer is reset to zero. This resetting occurs on the next CPU clock after the timer counter and the timer period match. Thus, the counter counts from zero to N. Consider the case where the period is 2 and the CPU clock/4 was selected as the timer clock source (CLKSRC = 1). Once started the timer would count, the following sequence: 0, 0, 0, 0, 1, 1, 1, 1, 2, 0, 0, 0, 1, 1, 1, 1, 2, 0, 0, 0…. Note that although the counter counts from 0 to 2, the period is 8 (= 2*4) CPU clock cycles rather than 12 (= 3*4) CPU clock cycles. Thus, the countdown period is TIMER PERIOD not TIMER PERIOD+1.

9.5 Timer Clock Source Selection: CLKSRC

Low to high transitions of the timer input clock allow the timer counter to increment. Two sources are available to drive the timer input clock:

1) The input value on the TINP pin. This signal is synchronized to prevent any metastability caused by asynchronous external inputs.

2) The CPU Clock/4.

9.6 Timer Pulse Generation

The two basic pulse generation modes are pulse mode and clock mode, as shown in Figure 9–5 and Figure 9–6, respectively. You can select the mode with the C/P- bit of the timer global control register. Note that in pulse mode, PWID in the timer control register can set the pulse width to either one or two input clock periods. The purpose of this feature is to provide minimum pulse widths in the case where TSTAT drives the TIM output. TSTAT drives this pin when TIM is used as a timer pin (FUNC = 1) and the CPU clock/4 is the clock source (CLKSRC = 0). Thus, the TIM pulse width is either 4 or 8 CPU clocks wide. Table 9–4 details equations for various TSTAT timing parameters in pulse and clock modes.

Figure 9–5. Timer Operation in Pulse Mode (CIP = 0)

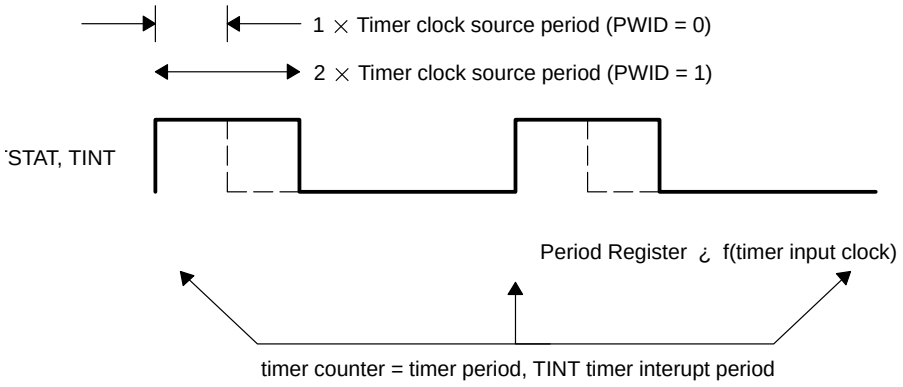


Figure 9–6. Timer Operation in Clock Mode (CIP = 1)

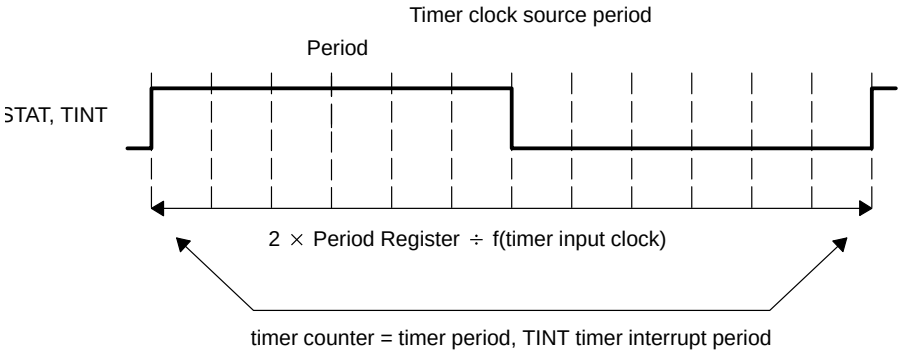


Table 9–4. TSTAT Parameters in Pulse and Clock Modes

Mode	Frequency	Period	Width High	Width Low
Pulse	$\frac{f \text{ (clock source)}}{\text{timer period register}}$	$\frac{\text{timer period register}}{f \text{ (clock source}}$	$\frac{\text{(PWID + 1)}}{f \text{ (clock source)}}$	$\frac{\text{timer period register – (PWID + 1)}}{f \text{ (clock source)}}$
Clock	$\frac{f \text{ (clock source)}}{\text{2 * timer period register}}$	$\frac{\text{2 * timer period register}}{f \text{ (clock source}}$	$\frac{\text{timer period register}}{f \text{ (clock source}}$	$\frac{\text{timer period register}}{f \text{ (clock source}}$

9.7 Boundary Conditions in the Control Registers

Certain boundary conditions affect timer operation:

1) Zero timer period register and counter register. After device reset and before the timer starts counting, TSTAT is held at 0. After the timer starts running by setting GO = 1, when the period and counter registers are zero, the operation of the timer depends on the C/P mode selected. In pulse mode, the TSTAT = 1 regardless whether or not it is held. In clock mode, when the timer is held ($\overline{HLD}$ = 0), TSTAT keeps it's previous value and when HLD = 1, TSTAT toggles with a frequency of (1/2 CPU clock frequency).

2) Counter overflow. When the counter register is set to a value greater than the value of the period register, the counter reaches its maximum value (FFFFFFFFh), rolls over to 0, and continues.

3) Writing to registers of an active timer. Writes from the peripheral bus override register updates to the counter register and new status updates to the control register.

4) Small timer period values in pulse mode. Note that small periods in pulse mode can cause TSTAT to remain high. This condition occurs when TIMER PERIOD ≤ PWID + 1.

9.8 Timer Interrupts

The TSTAT signal directly drives the CPU interrupt as well as a DMA synchronization event. The frequency of the interrupt is the same as the frequency of the TSTAT.

9.9 Emulation Operation

During debug using the emulator, the CPU may be halted on an execute packet boundary for single stepping, benchmarking, profiling, or other debug uses. During an emulation halt, the timer halts when the CPU clock/4 is selected as the clock source (CLKSRC = 1). Here, the counter is only enabled to count during those cycles when the CPU is not stalled due to the emulation halt. Thus, counting will be re-enabled during single-step operation. If CLKSRC = 0, the timer continues counting as programmed.

Chapter 10

Interrupt Selector and External Interrupts

This chapter describes the interrupt selector and registers available for the TMS320C62xx devices.

Topic **Page**

10.1 Overview

The ‘C6x peripheral set produces 16 interrupt sources. The CPU however has 12 interrupts available for use. The interrupt selector allows you to choose and prioritize which 12 of the 16 your system needs to use. The interrupt selector also allows you to effectively change the polarity of external interrupt inputs.

10.2 Available Interrupt Sources

Table 10–1 lists the available interrupts. Note that this table is similar to the DMA synchronization events in the Chapter 4 DMA Controller except for two differences. One difference is that the MCSP generates separate interrupts and DMA synchronization events. The second difference is that DSPINT has been moved to allow a 4-bit encoding.

Table 10–1. Available Interrupts

Interrupt Selection Number	Interrupt Acronym	Interrupt Description
0000b	DSPINT	Host Port Host to DSP Interrupt
0001b	TINT0	Timer 0 Interrupt
0010b	TINT1	Timer 1 Interrupt
0011b	SD_INT	EMIF SDRAM Timer Interrupt
0100b	EXT_INT4	External Interrupt Pin 4
0101b	EXT_INT5	External Interrupt Pin 5
0110b	EXT_INT6	External Interrupt Pin 6
0111b	EXT_INT7	External Interrupt Pin 7
1000b	DMA_INT0	DMA Channel 0 Interrupt
1001b	DMA_INT1	DMA Channel 1 Interrupt
1010b	DMA_INT2	DMA Channel 2 Interrupt
1011b	DMA_INT3	DMA Channel 3 Interrupt
1100b	XINT0	MCSP 0 Transmit Interrupt
1101b	RINT0	MCSP 0 Receive Interrupt
1110b	XINT1	MCSP 1 Transmit Interrupt
1111b	RINT1	MCSP 1 Receive Interrupt

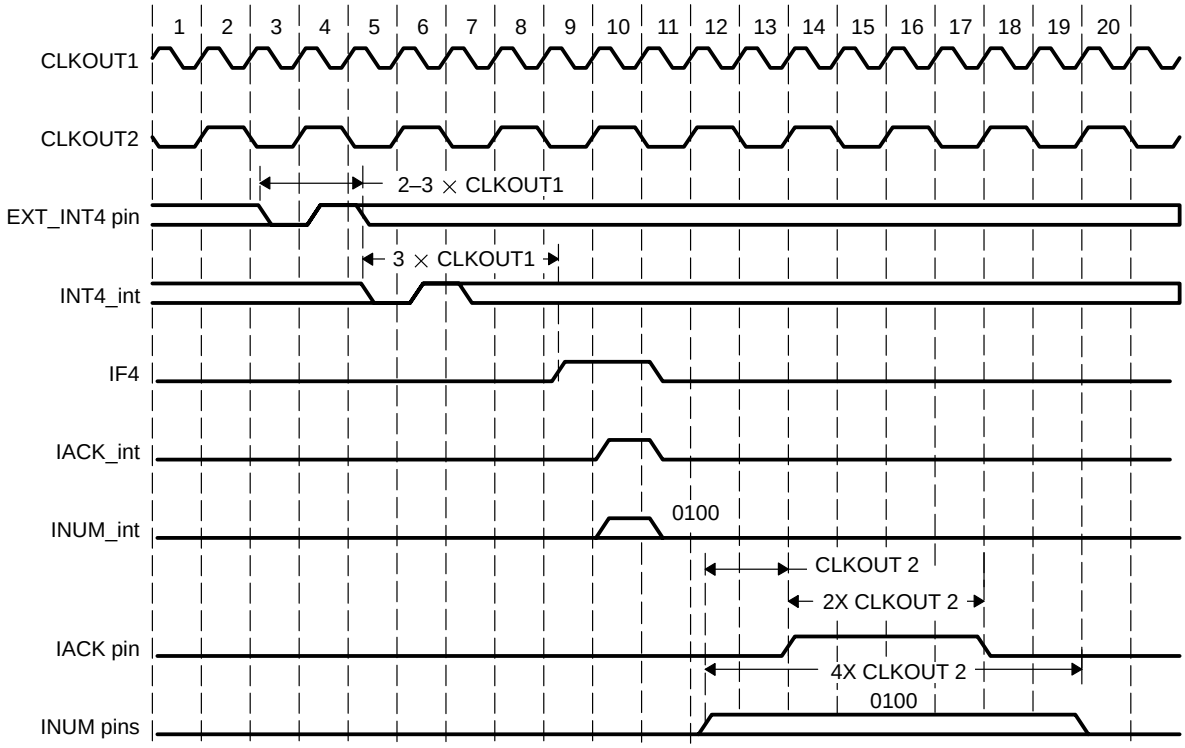
For more information on interrupts, including the interrupt vector table, see the *TMS320C62xx CPU and Instruction Set Reference Guide*.

10.3 External Interrupt Signal Timing

EXT_ and NMI are dedicated external interrupt sources. In addition, the FSR and FSX can be programmed to directly drive the RINT and XINT signals. Because these signals are asynchronous, they are passed through two registers before being sent to either the DMA or CPU. Figure 10–1 shows the timing of external interrupt signals using INT4 as an example. This diagram is similar to the one in the CPU Reference Guide. However, this diagram also shows the delays for the external interrupt through the two synchronization flip-flops. Note, that this delay is two CPU clock (CLKOUT1) cycles. However, if the INT4 input transitions during the setup and hold time with respect to the CLKOUT1 rising edge, this delay could be as long as 3 CLKOUT1 cycles. Once synchronized, an additional 3 CLKOUT1 cycle delay occurs before the related interrupt flag (IF4) is set.

The interrupt can only be scheduled to be taken one CLKOUT1 cycle later at the earliest as indicated by the active internal interrupt acknowledge (IACK) signal. The interrupt can be postponed or inhibited if not properly enabled as described in other chapters of the CPU Reference Guide. In that case, IACK will be also be postponed. Along with IACK, the CPU sets the INUM signal to indicate which interrupt was taken. Externally, the IACK pin pulse is extended to two CLKOUT2 cycles wide and synchronized to CLKOUT2. Also, the INUM pin signal frames this external IACK with one CLKOUT2 cycle of setup and hold, for a width of 4 CLKOUT2 cycles. Note that even though INUM and IACK in the diagram are not valid on a CLKOUT2 rising edge, the internal circuitry still catches the transition and produces the desired waveforms on the IACK and INUM pins.

Figure 10–1. Timing of External Interrupt Related Signals



10.4 Interrupt Selector Registers

Table 10–2 shows the interrupt selector registers. The Interrupt Multiplexor Registers determine the mapping between the interrupt sources in Table 10–1 and the CPU interrupts 4 through 15 (INT4-INT15). The External Interrupt Polarity register sets the polarity of external interrupts.

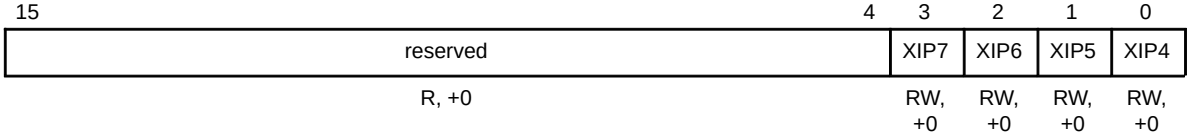
Table 10–2. Interrupt Selector Registers

Hex Byte Address	Name	Description	Section
019C0000	Interrupt Multiplexer High	Selects which interrupts drive CPU interrupts 10–15 (INT10–15)	10.4.2
019C0004	Interrupt Multiplexer Low	Selects which interrupts drive CPU interrupts 4–9 (INT4–INT9)	10.4.2
019C0008	External Interrupt Polarity	Sets the polarity of the external interrupts (EXT_INT4–EXT_INT7)	10.4.1

10.4.1 External Interrupt Polarity Register

The external interrupt polarity register allows you to change the polarity of the four external interrupts (EXT_INT4-EXT_INT7). Normally, a low-to-high transition on an interrupt source is recognized as an interrupt. By setting the related XIP bit in this register to 1, you can invert the external interrupt source and effectively have the CPU detect high-to-low transitions of the external interrupt. If the related XIP is cleared to 0, then the non-inverted external interrupt is passed and the CPU recognizes a low-to-high transition as signaling an interrupt.

Figure 10–2. External Interrupt Polarity Register



10.4.2 Interrupt Multiplexer Register

The INTSEL fields in the Interrupt Multiplexer Registers allow mapping the interrupt sources in to particular interrupts. The INTSEL4-INTSEL15 correspond to CPU interrupts INT4-INT15 as shown in Table 10–3. By setting the INTSEL fields to the value of the desired interrupt selection number in Table 10–2, you may map any interrupt source to any CPU interrupt. Table 10–3 also shows the default mapping of interrupt sources to CPU interrupts.

Figure 10–3. Interrupt Multiplexer Low Register Diagram

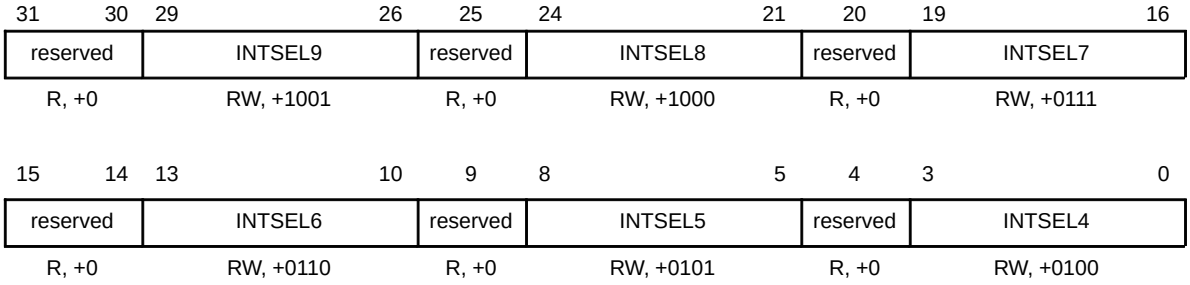


Figure 10–4. Interrupt Multiplexer High Register Diagram

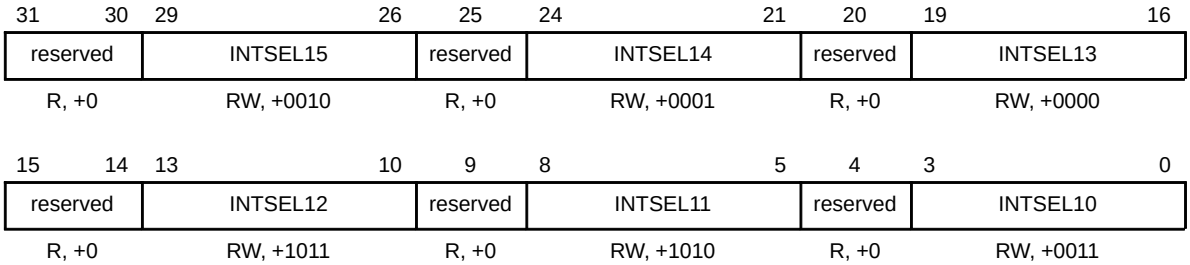


Table 10–3. Default Interrupt Mapping

CPU Interrupt	Related INTSEL field	INTSEL Reset Value	Interrupt Acronym	Interrupt Description
INT4	INTSEL4	0100	EXT_INT4	External Interrupt Pin 4
INT5	INTSEL5	0101	EXT_INT5	External Interrupt Pin 5
INT6	INTSEL6	0110	EXT_INT6	External Interrupt Pin 6
INT7	INTSEL7	0111	EXT_INT7	External Interrupt Pin 7
INT8	INTSEL8	1000	DMA_INT0	DMA Channel 0 Interrupt
INT9	INTSEL9	1001	DMA_INT1	DMA Channel 1 Interrupt
INT10	INTSEL10	0011	SD_INT	EMIF SDRAM Timer Interrupt
INT11	INTSEL11	1010	DMA_INT2	DMA Channel 2 Interrupt
INT12	INTSEL12	1011	DMA_INT3	DMA Channel 3 Interrupt
INT13	INTSEL13	0000	DSPINT	Host Port Host to DSP Interrupt
INT14	INTSEL14	0001	TINT0	Timer 0 Interrupt
INT15	INTSEL15	0010	TINT1	Timer 1 Interrupt

10.5 Configuring the Interrupt Selector

The interrupt selector registers is meant to be configured once after reset during initialization before enabling interrupts. Once the registers have been set, the interrupt flag register should be cleared by the user after some delay to remove any spurious transitions caused by the configuration. You may reconfigure the interrupt selector during other times, but spurious interrupt conditions maybe detected by the CPU on the interrupts affected by the modified fields.

Chapter 11

Device Clocking

This chapter describes the PLL device clocking for the TMS320C62xx.

Topic **Page**

11.1 Overview

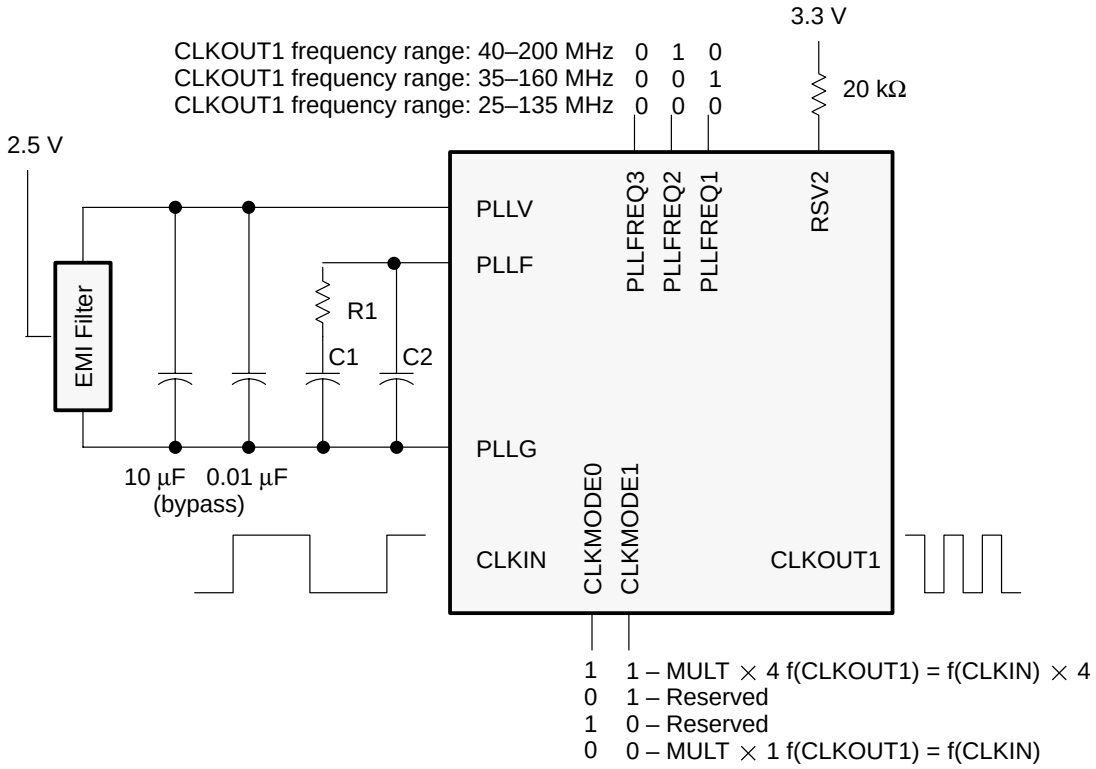
An external oscillator drives the on-chip PLL (Phase-Locked Loop) circuit that generates all internal and external clocks. The PLL multiplies the external oscillator frequency by 4 and feeds the resulting clock to CLKOUT1 output pin. The internal version of CLKOUT1 is used by the processor as an instruction cycle clock. Most timing parameters of this device are defined relative to the CLKOUT1 clock and specifically to it's rising edge. CLKOUT2 is another output clock derived in EMIF from CLKOUT1 at half of its frequency. Two other clock signals are derived from the PLL outputs – SSCLK and SDCLK (also in EMIF). They are used to clock the external SBSRAM and SDRAM synchronous memories.

In addition x4 mode, the clock circuit can operate in multiply by 1 mode, where the input clock frequency is the same as the CLKOUT1 output clock frequency. The factors to consider in choosing the multiply factor include board level noise and clock jitter. The x4 mode will minimize the board noise, while the x1 mode will reduce the internal clock jitter. The clock mode is controlled by two CLKMODE pins as shown in the following Figure.

The amount of time that the PLL needs to synchronize to the output frequency depends on the CLKIN and CLKOUT1 frequencies and is typically in the range of tens of microseconds. See the TMS320C6201 device data sheet table for the exact time. The synchronization time affects the duration of the Reset signal in that the reset has to be asserted long enough for the PLL to synchronize to the proper output frequency.

Three PLLFREQ pins identify the range of CLKOUT1 frequencies that the PLL is expected to synchronize to. The PLL also requires 2 bypass capacitors (between PLLV and PLLG), external low-pass filter components (R1, C1 ,C2) and an EMI filter. The values for R1, C1, C2 and the filter depend on the CLKIN and CLKOUT1 frequencies. Refer to the Data Sheet for PLL external component and the EMI filter values.

Figure 11–1. Clock PLL DIagram



- For CLKMODE x4, values for C1, C2 and R2 depend on CLKIN and CLKOUT frequencies.
- For CLKMODE x1, the PLL is by-passed and all six external PLL components can be removed. For this case, the PLLV terminal has to be connected to a clean 2.5V supply and the PLLG and PLLF terminals should be tied together.
- Due to overlap of frequency ranges when choosing the PLLFREQ more that one frequency range can contain the desired CLKOUT1 frequency. Choose the lowest frequency range including the desired frequency. For example, for CLKOUT1 = 133MHz, chose PLLFREQ value of 000b. For CLKOUT1 = 166 or 200MHz choose PLLFREQ value of 010b. PLLFREQ values other then 000b, 001b and 010b are reserved.

Chapter 12

Power-Down Logic

This chapter describes the power-down modes of the TMS320C62xx.

Topic **Page**

12.1 Overview

Most of the operating power of CMOS logic is dissipated during circuit switching from one logic state to another. By preventing some or all of chip's logic from switching, significant power savings can be realized without losing any data or operational context. Power-down mode PD1 blocks the internal clock inputs at the boundary of the CPU, preventing most of its logic from switching. PD1 effectively shuts down the CPU. Additional power savings are accomplished in power-down mode PD2, where the entire on-chip clock structure (including multiple buffers) is "halted" at the output of the PLL (see Figure 12–1 below). PD3 is like PD2 but also disconnects the external clock source (CLKIN) from reaching the PLL. Wake-up from PD3 takes longer then wake-up from PD2 because the PLL needs to be re-locked, just as it does following power-up.

PD2 and PD3 halt the entire chip in both PLL clock modes - x1 and x4. Both the PD2 and PD3 signals also assert the $\overline{PD}$ pin for external recognition of these two power-down modes. In addition to power-down modes described in this chapter, the IDLE instruction provides lower CPU power consumption by executing multiple NOPs. The IDLE instruction terminates only upon servicing an interrupt.In addition, the reset condition forces a complete pwer down.

Figure 12–1. Power-Down Mode Logic

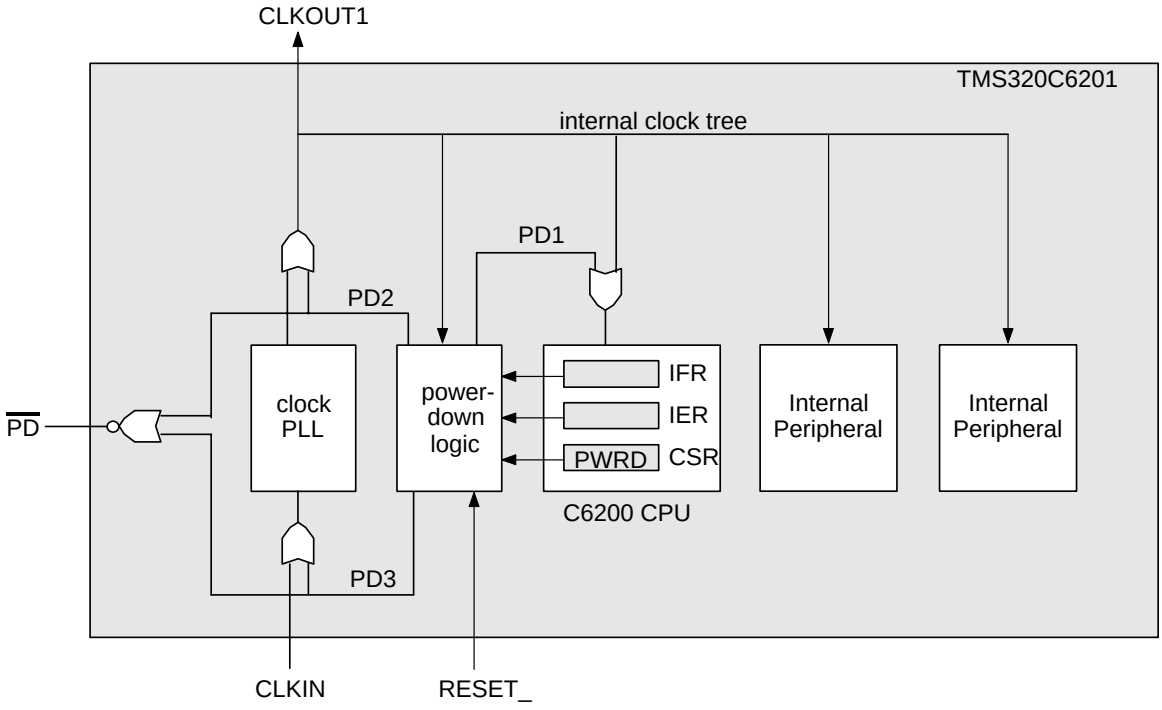
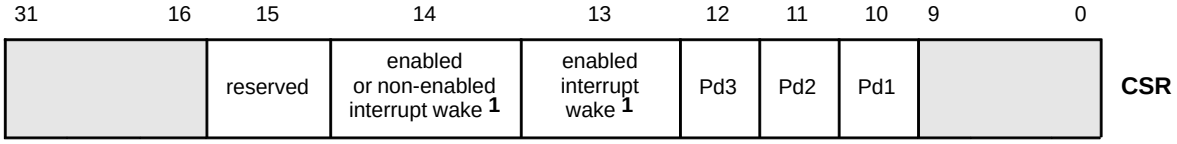


Figure 12–2. PRWD Field of the CSR Register



The power-down modes and their wake-up methods are programmed by setting bits 10-15 in the of the Control Status Register (CSR PWRD field). PD2 and PD3 modes can only be aborted by device Reset, while PD1 mode can also be terminated by an enabled interrupt, or any interrupt (enabled or not), as directed by bits 13 and 14 of the CSR (interrupts are edge driven). When writing to CSR, all bits of the PWRD field should be set at the same time. Logic 0 should be used when writing to reserved fields (bit 15 of CSR).

Table 12–1. Power-Down Mode and Wake-Up Selection

PRWD	Power-down mode/Wake-up method
000000	no power-down
001001	Pd1 / wake by an enabled interrupt
010001	Pd1 / wake by an enabled or non-enabled interrupt
011010	Pd2
011100	Pd3
	reserved

12.2 Triggering, Wake-up, and Effects

Power-down mode PD1 takes effect 3-4 instructions after the instruction that caused the power-down (by setting the idle bits in the CSR). See the instruction flow below:

INSTR1 (MVC)	power-down mode is set by this instruction
INSTR2	CPU notifies power-down logic to initiate power-down
INSTR3	power-down signal is sent to CPU and/or peripherals
INSTR4	CPU receives the internal power-down signal
INSTR5	CPU suspends execution before INSTR5 or INSTR6
INSTR6	CPU resumes execution with INSTR5 or INSTR6
INSTR7	normal program execution resumed here

The wake-up from Pd1 can be triggered by either an enabled interrupt, or any interrupt (enabled or not). The first case is selected by writing a logic 1 to bit 13 of the Control Status Register (PWRD field), and the second case is selected by writing a logic 1 into bit 14 of CSR. If PD1 mode is terminated by a non-enabled interrupt, the program execution returns to the 4^{th} or 5^{th} instruction following the one that caused the power-down (by setting the idle bits in the CSR). Wake-up by an enabled interrupt executes the corresponding Interrupt Service Fetch Packet first, prior to returning to the 4^{th} or 5^{th} instruction following the one that caused the power-down (CSR register GIE bit and IER register NMIE bit must also be set in order for the ISFP to execute), otherwise execution returns to the previous point.

Table 12–2. Characteristics of the Power-Down Modes

Power-down Mode	Trigger Action	Wake-up Method	Effect on Chip's Operation
PD1	write logic 001001b or 010001b to bits 15-10 of the CSR	internal interrupt, external interrupt or Reset	CPU halted (except for the interrupt logic)
PD2	write logic 011010b to bits 15-10 of the CSR	Reset only	Output clock from PLL is halted, stopping the internal clock structure from switching and resulting in the entire chip being halted. Signal terminal PD_ is driven low. All register and internal RAM contents are preserved. All signal terminals behave the same way as during Reset.
PD3	write logic 11100b to bits 15-10 of the CSR	Reset only	Input clock to the PLL is halted, shutting down the PLL and stopping the internal clock structure from switching and resulting in the entire chip being halted. Signal terminal PD_ is driven low. All register and internal RAM contents are preserved. All signal terminals behave the same way as during Reset. Following reset, the PLL needs time to re-lock, just as it does following power-up.

Chapter 13

Designing for JTAG Emulation

This chapter assists you in meeting the design requirements of the XDS510 emulator with respect to JTAG designs and discusses the XDS510 cable (manufacturing part number 2617698-0001). This cable is identified by a label on the cable pod marked **JTAG 3/5V** and supports both standard 3-volt and 5-volt target system power inputs.

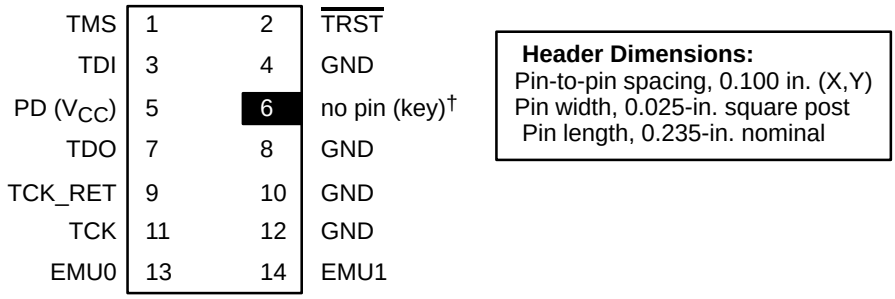
The term *JTAG,* as used in this book, refers to TI scan-based emulation, which is based on the IEEE 1149.1 standard.

Topic **Page**

13.1 Designing Your Target System's Emulator Connector (14-Pin Header)

JTAG target devices support emulation through a dedicated emulation port. This port is a superset of the IEEE 1149.1 standard and is accessed by the emulator. To communicate with the emulator, **your target system must have a 14-pin header** (two rows of seven pins) with the connections that are shown in Figure 13–1. Table 13–1 describes the emulation signals.

Figure 13–1. 14-Pin Header Signals and Header Dimensions



† While the corresponding female position on the cable connector is plugged to prevent improper connection, the cable lead for pin 6 is present in the cable and is grounded, as shown in the schematics and wiring diagrams in this document.

Table 13–1. 14-Pin Header Signal Descriptions

Signal	Description	Emulator† State	Target† State
TMS	Test mode select	O	I
TDI	Test data input	O	I
TDO	Test data output	I	O
TCK	Test clock. TCK is a 10.368-MHz clock source from the emulation cable pod. This signal can be used to drive the system test clock	O	I
$\overline{\text{TRST}}$‡	Test reset	O	I
EMU0	Emulation pin 0	I	I/O
EMU1	Emulation pin 1	I	I/O
PD(V_{CC})	Presence detect. Indicates that the emulation cable is connected and that the target is powered up. PD should be tied to V_{CC} in the target system.	I	O
TCK_RET	Test clock return. Test clock input to the emulator. May be a buffered or unbuffered version of TCK.	I	O
GND	Ground		

† I = input; O = output

‡ Do not use pullup resistors on $\overline{\text{TRST}}$: it has an internal pulldown device. In a low-noise environment, $\overline{\text{TRST}}$ can be left floating. In a high-noise environment, an additional pulldown resistor may be needed. (The size of this resistor should be based on electrical current considerations.)

Although you can use other headers, recommended parts include:

straight header, unshrouded	DuPont Connector Systems part numbers:	65610–114
		65611–114
		67996–114
		67997–114

13.2 Bus Protocol

The IEEE 1149.1 specification covers the requirements for the test access port (TAP) bus slave devices and provides certain rules, summarized as follows:

- The TMS/TDI inputs are sampled on the rising edge of the TCK signal of the device.
- The TDO output is clocked from the falling edge of the TCK signal of the device.

When these devices are daisy-chained together, the TDO of one device has approximately a half TCK cycle setup to the next device's TDI signal. This type of timing scheme minimizes race conditions that would occur if both TDO and TDI were timed from the same TCK edge. The penalty for this timing scheme is a reduced TCK frequency.

The IEEE 1149.1 specification does not provide rules for bus master (emulator) devices. Instead, it states that it expects a bus master to provide bus slave compatible timings. The XDS510 provides timings that meet the bus slave rules.

13.3 IEEE 1149.1 Standard

For more information concerning the IEEE 1149.1 standard, contact IEEE Customer Service:

Address: IEEE Customer Service
445 Hoes Lane, PO Box 1331
Piscataway, NJ 08855-1331

Phone: (800) 678–IEEE in the US and Canada
(908) 981–1393 outside the US and Canada

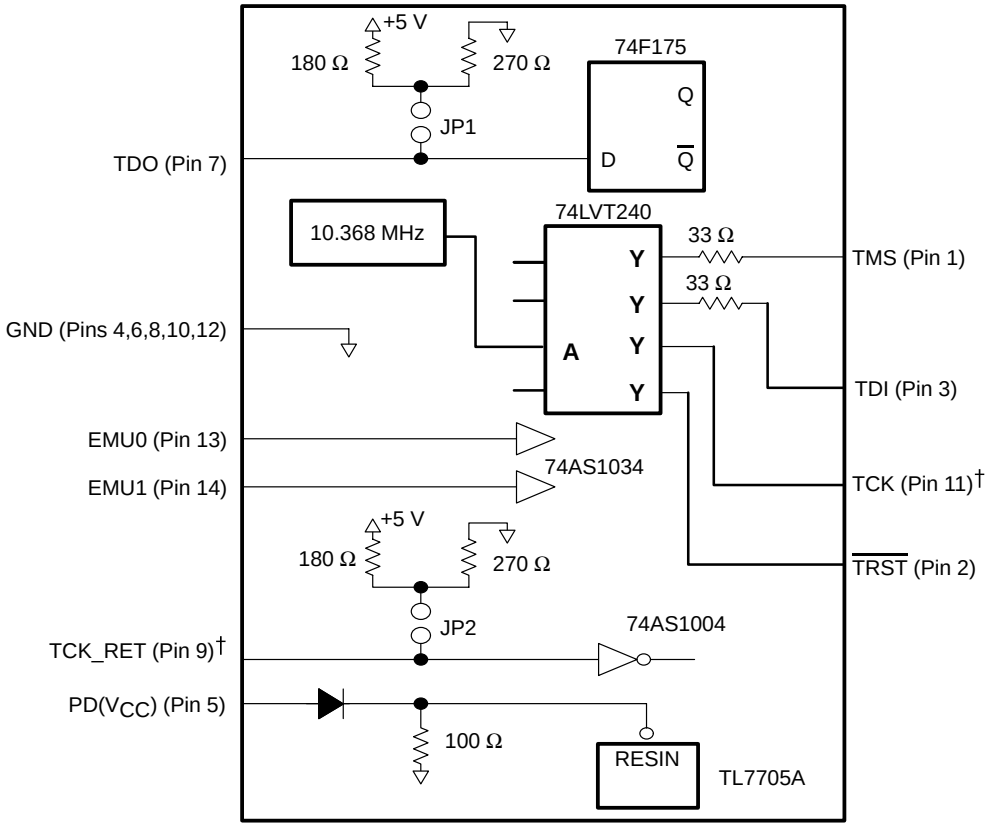
FAX: (908) 981–9667 Telex: 833233

13.4 JTAG Emulator Cable Pod Logic

Figure 13–2 shows a portion of the emulator cable pod. These are the functional features of the pod:

- Signals TDO and TCK_RET can be parallel-terminated inside the pod if required by the application. By default, these signals are not terminated.
- Signal TCK is driven with a 74LVT240 device. Because of the high-current drive (32 mA I_{OL}/I_{OH}), this signal can be parallel-terminated. If TCK is tied to TCK_RET, then you can use the parallel terminator in the pod.
- Signals TMS and TDI can be generated from the falling edge of TCK_RET, according to the IEEE 1149.1 bus slave device timing rules.
- Signals TMS and TDI are series-terminated to reduce signal reflections.
- A 10.368-MHz test clock source is provided. You may also provide your own test clock for greater flexibility.

Figure 13–2. JTAG Emulator Cable Pod Interface



† The emulator pod uses TCK_RET as its clock source for internal synchronization. TCK is provided as an optional target system test clock source.

13.5 JTAG Emulator Cable Pod Signal Timing

Figure 13–3 shows the signal timings for the emulator cable pod. Table 13–2 defines the timing parameters. These timing parameters are calculated from values specified in the standard data sheets for the emulator and cable pod and are for reference only. Texas Instruments does not test or guarantee these timings.

The emulator pod uses TCK_RET as its clock source for internal synchronization. TCK is provided as an optional target system test clock source.

Figure 13–3. JTAG Emulator Cable Pod Timings

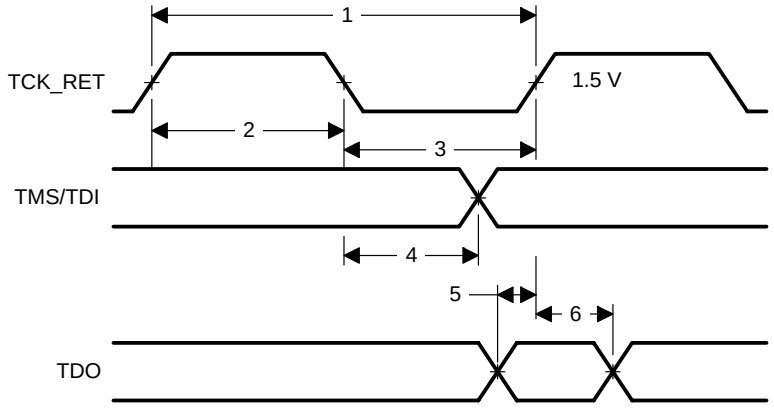


Table 13–2. Emulator Cable Pod Timing Parameters

No.	Reference	Description	Min	Max	Units
1	$t_{c(TCK)}$	TCK_RET period	35	200	ns
2	$t_{w(TCKH)}$	TCK_RET high-pulse duration	15		ns
3	$t_{w(TCKL)}$	TCK_RET low-pulse duration	15		ns
4	$t_{d(TMS)}$	Delay time, TMS/TDI valid from TCK_RET low	6	20	ns
5	$t_{su(TDO)}$	TDO setup time to TCK_RET high	3		ns
6	$t_{h(TDO)}$	TDO hold time from TCK_RET high	12		ns

13.6 Emulation Timing Calculations

The following examples help you calculate emulation timings in your system. For actual target timing parameters, see the appropriate device data sheets.

Assumptions:

$t_{su(TTMS)}$	Target TMS/TDI setup to TCK high	10 ns
$t_{d(TTDO)}$	Target TDO delay from TCK low	15 ns
$t_{d(bufmax)}$	Target buffer delay, maximum	10 ns
$t_{d(bufmin)}$	Target buffer delay, minimum	1 ns
$t_{(bufskew)}$	Target buffer skew between two devices in the same package: $[t_{d(bufmax)} - t_{d(bufmin)}] \times 0.15$	1.35 ns
$t_{(TCKfactor)}$	Assume a 40/60 duty cycle clock	0.4 (40%)

Given in Table 13–2 (on page 13-6):

$t_{d(TMSmax)}$	Emulator TMS/TDI delay from TCK_RET low, maximum	20 ns
$t_{su(TDOmin)}$	TDO setup time to emulator TCK_RET high, minimum	3 ns

There are two key timing paths to consider in the emulation design:

- The TCK_RET-to-TMS/TDI path, called $t_{pd(TCK\_RET–TMS/TDI)}$, and
- The TCK_RET-to-TDO path, called $t_{pd(TCK\_RET–TDO)}$.

Of the following two cases, the worst-case path delay is calculated to determine the maximum system test clock frequency.

Case 1: Single processor, direct connection, TMS/TDI timed from TCK_RET low.

$$t_{pd\,(TCK\_RET-TMS/TDI)} = \frac{\left[t_{d\,(TMSmax)} + t_{su\,(TTMS)}\right]}{t_{(TCKfactor)}}$$
$$= \frac{[20ns + 10ns]}{0.4}$$
$$= 75ns\ (13.3\ MHz)$$
$$t_{pd\,(TCK\_RET-TDO)} = \frac{\left[t_{d\,(TTDO)} + t_{su\,(TDOmin)}\right]}{t_{(TCKfactor)}}$$
$$= \frac{[15ns + 3ns]}{0.4}$$
$$= 45ns\ (22.2\ MHz)$$

In this case, the TCK_RET-to-TMS/TDI path is the limiting factor.

Case 2: Single/multiprocessor, TMS/TDI/TCK buffered input, TDO buffered output, TMS/TDI timed from TCK_RET low.

$$t_{pd\,(TCK\_RET-TMS/TDI)} = \frac{\left[t_{d\,(TMSmax)} + t_{su\,(TTMS)} + t_{(bufskew)}\right]}{t_{(TCKfactor)}}$$
$$= \frac{[20ns + 10ns + 1.35ns]}{0.4}$$
$$= 78.4ns\ (12.7\ MHz)$$
$$t_{pd\,(TCK\_RET-TDO)} = \frac{\left[t_{d\,(TTDO)} + t_{su\,(TDOmin)} + t_{d\,(bufmax)}\right]}{t_{(TCKfactor)}}$$
$$= \frac{[15ns + 3ns + 10ns]}{0.4}$$
$$= 70ns\ (14.3\ MHz)$$

In this case, the TCK_RET-to-TMS/TDI path is the limiting factor.

In a multiprocessor application, it is necessary to ensure that the EMU0–1 lines can go from a logic low level to a logic high level in less than 10 μs. This can be calculated as follows:

$$
\begin{aligned}
t_r &= 5(R_{pullup} \times N_{devices} \times C_{load\_per\_device}) \\
&= 5(4.7\ k\Omega \times 16 \times 15\ pF) \\
&= 5.64\ \mu s
\end{aligned}
$$

13.7 Connections Between the Emulator and the Target System

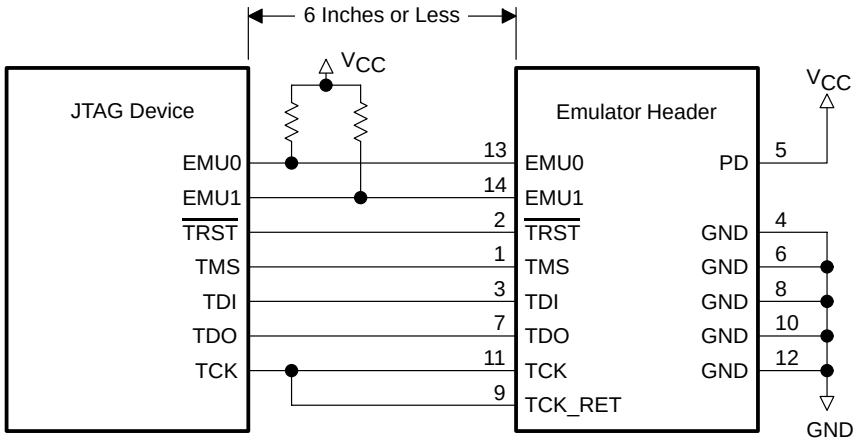
It is extremely important to provide high-quality signals between the emulator and the JTAG target system. Depending upon the situation, you must supply the correct signal buffering, test clock inputs, and multiple processor interconnections to ensure proper emulator and target system operation.

Signals applied to the EMU0 and EMU1 pins on the JTAG target device can be either input or output (I/O). In general, these two pins are used as both input and output in multiprocessor systems to handle global run/stop operations. EMU0 and EMU1 signals are applied only as inputs to the XDS510 emulator header.

13.7.1 Buffering Signals

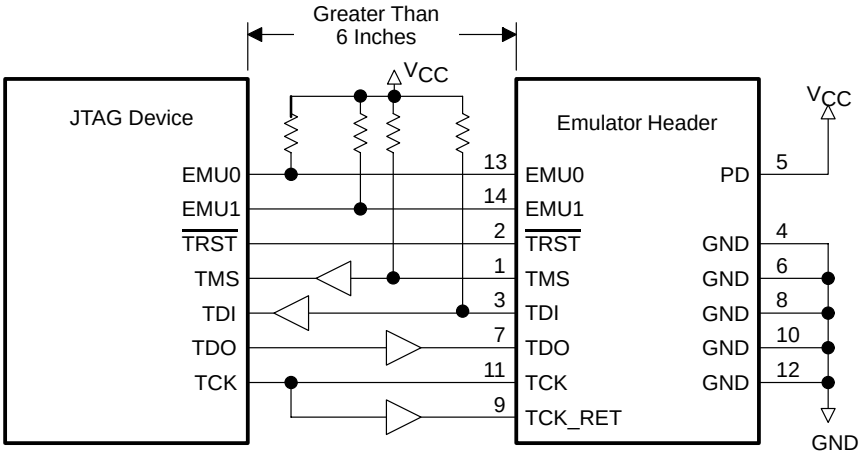
If the distance between the emulation header and the JTAG target device is greater than six inches, the emulation signals must be buffered. If the distance is less than six inches, no buffering is necessary. The following illustrations depict these two situations.

- **No signal buffering.** In this situation, the distance between the header and the JTAG target device should be no more than six inches.



The EMU0 and EMU1 signals must have pullup resistors connected to V_{CC} to provide a signal rise time of less than 10 μs. A 4.7-kΩ resistor is suggested for most applications.

❑ **Buffered transmission signals.** In this situation, the distance between the emulation header and the processor is greater than six inches. Emulation signals TMS, TDI, TDO, and TCK_RET are buffered through the same package.



■ The EMU0 and EMU1 signals must have pullup resistors connected to V_{CC} to provide a signal rise time of less than 10 µs. A 4.7-kΩ resistor is suggested for most applications.

■ The input buffers for TMS and TDI should have pullup resistors connected to V_{CC} to hold these signals at a known value when the emulator is not connected. A resistor value of 4.7 kΩ or greater is suggested.

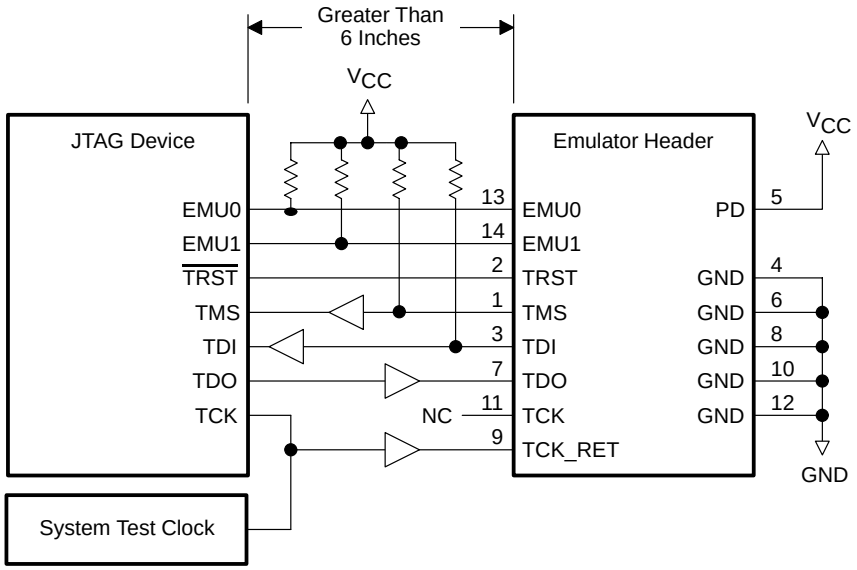
■ To have high-quality signals (especially the processor TCK and the emulator TCK_RET signals), you may have to employ special care when routing the PWB trace. You also may have to use termination resistors to match the trace impedance. The emulator pod provides optional internal parallel terminators on the TCK_RET and TDO. TMS and TDI provide fixed series termination.

■ Since $\overline{\text{TRST}}$ is an asynchronous signal, it should be buffered as needed to insure sufficient current to all target devices.

13.7.2 Using a Target-System Clock

Figure 13–4 shows an application with the system test clock generated in the target system. In this application, the TCK signal is left unconnected.

Figure 13–4. Target-System-Generated Test Clock



Note: When the TMS/TDI lines are buffered, pullup resistors should be used to hold the buffer inputs at a known level when the emulator cable is not connected.

There are two benefits to having the target system generate the test clock:

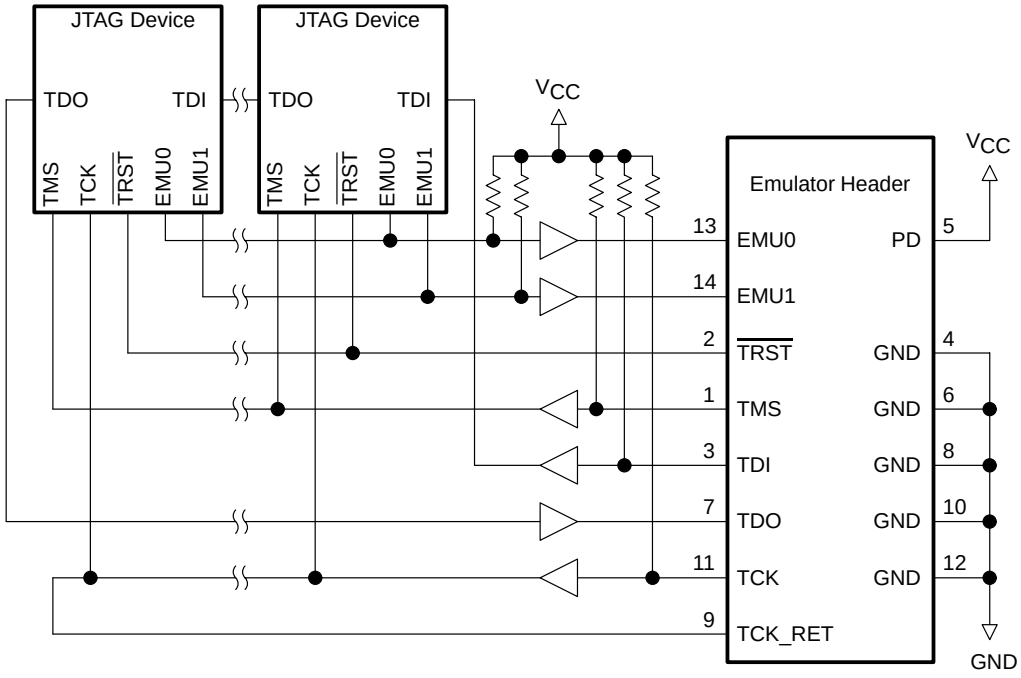
- The emulator provides only a single 10.368-MHz test clock. If you allow the target system to generate your test clock, you can set the frequency to match your system requirements.
- In some cases, you may have other devices in your system that require a test clock when the emulator is not connected. The system test clock also serves this purpose.

13.7.3 Configuring Multiple Processors

Figure 13–5 shows a typical daisy-chained multiprocessor configuration, which meets the minimum requirements of the IEEE 1149.1 specification. The emulation signals in this example are buffered to isolate the processors from the emulator and provide adequate signal drive for the target system. One of the benefits of this type of interface is that you can generally slow down the test clock to eliminate timing problems. You should follow these guidelines for multiprocessor support:

- ❑ The processor TMS, TDI, TDO, and TCK signals should be buffered through the same physical package for better control of timing skew.
- ❑ The input buffers for TMS, TDI, and TCK should have pullup resistors connected to V_{CC} to hold these signals at a known value when the emulator is not connected. A resistor value of 4.7 kΩ or greater is suggested.
- ❑ Buffering EMU0 and EMU1 is optional but highly recommended to provide isolation. These are not critical signals and do not have to be buffered through the same physical package as TMS, TCK, TDI, and TDO. Unbuffered and buffered signals are shown in this section (page 13-10 and page 13-11).

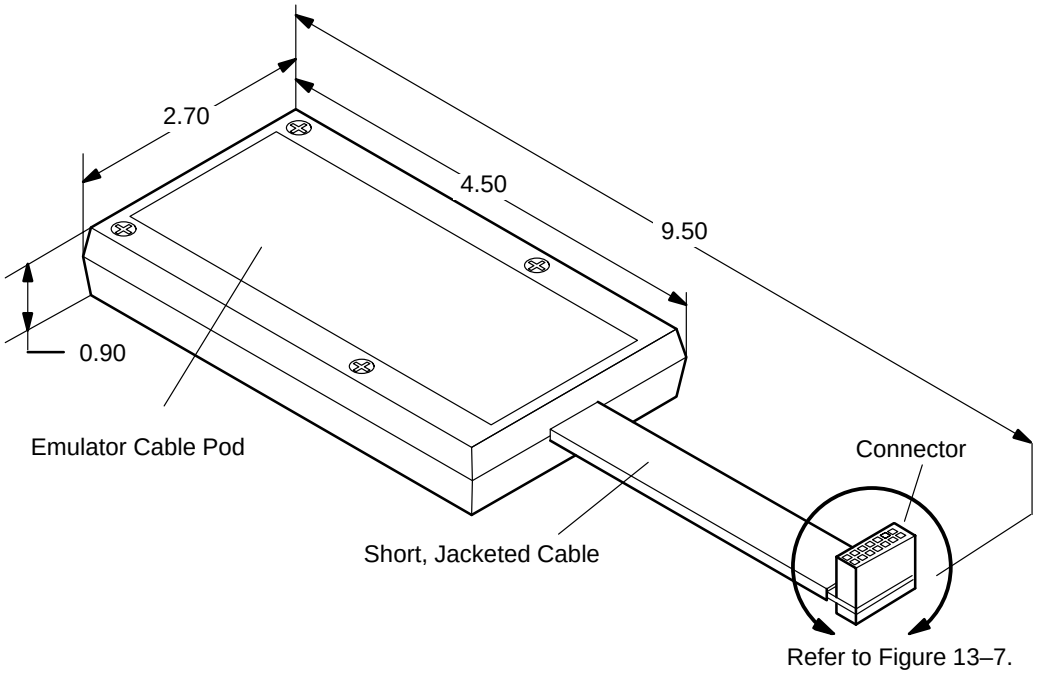
Figure 13–5. Multiprocessor Connections



13.8 Mechanical Dimensions for the 14-Pin Emulator Connector

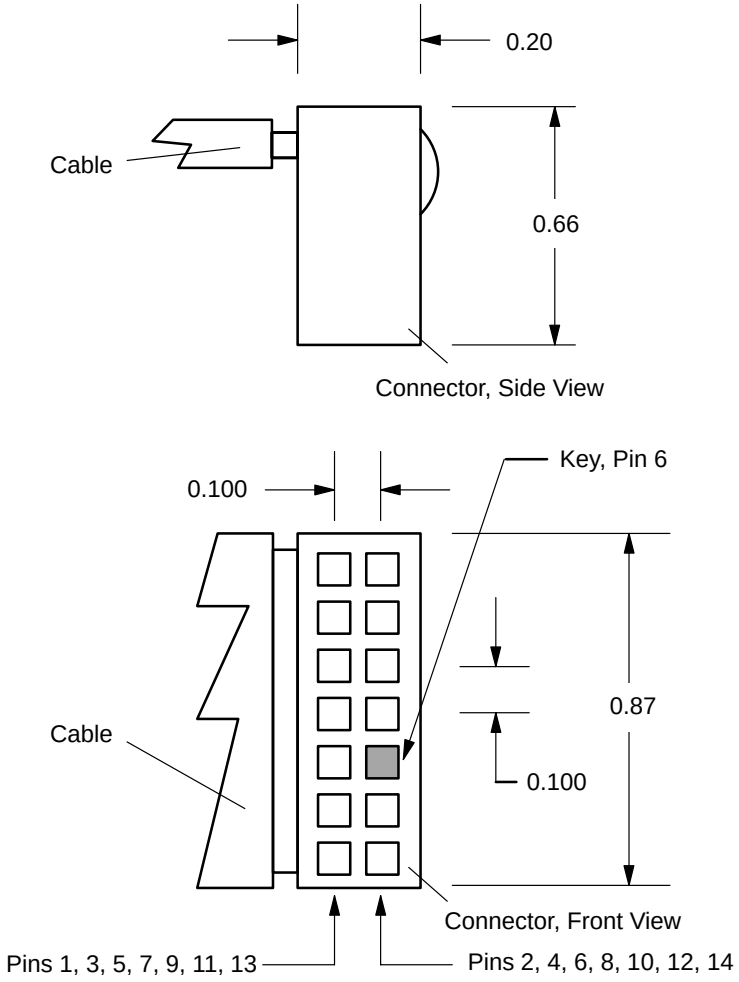
The JTAG emulator target cable consists of a 3-foot section of jacketed cable, an active cable pod, and a short section of jacketed cable that connects to the target system. The overall cable length is approximately 3 feet 10 inches. Figure 13–6 and Figure 13–7 (page 13-15) show the mechanical dimensions for the target cable pod and short cable. Note that the pin-to-pin spacing on the connector is 0.100 inches in both the X and Y planes. The cable pod box is nonconductive plastic with four recessed metal screws.

Figure 13–6. Pod/Connector Dimensions



Note: All dimensions are in inches and are nominal dimensions, unless otherwise specified.

Figure 13–7. 14-Pin Connector Dimensions



Note: All dimensions are in inches and are nominal dimensions, unless otherwise specified.

13.9 Emulation Design Considerations

This section describes the use and application of the scan path linker (SPL), which can simultaneously add all four secondary JTAG scan paths to the main scan path. It also describes the use of the emulation pins and the configuration of multiple processors.

13.9.1 Using Scan Path Linkers

You can use the TI ACT8997 scan path linker (SPL) to divide the JTAG emulation scan path into smaller, logically connected groups of 4 to 16 devices. As described in the *Advanced Logic and Bus Interface Logic Data Book* (literature number SCYD001), the SPL is compatible with the JTAG emulation scanning. The SPL is capable of adding any combination of its four secondary scan paths into the main scan path.

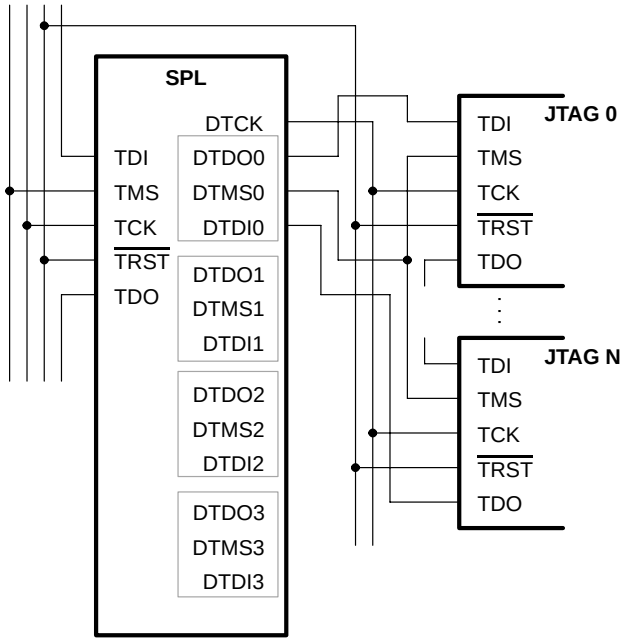
A system of multiple, secondary JTAG scan paths has better fault tolerance and isolation than a single scan path. Since an SPL has the capability of adding all secondary scan paths to the main scan path simultaneously, it can support global emulation operations, such as starting or stopping a selected group of processors.

TI emulators do not support the nesting of SPLs (for example, an SPL connected to the secondary scan path of another SPL). However, you can have multiple SPLs on the main scan path.

Although the ACT8999 scan path selector is similar to the SPL, it can add only one of its secondary scan paths at a time to the main JTAG scan path. Thus, global emulation operations are not assured with the scan path selector. For this reason, scan path selectors are not supported.

You can insert an SPL on a backplane so that you can add up to four device boards to the system without the jumper wiring required with nonbackplane devices. You connect an SPL to the main JTAG scan path in the same way you connect any other device. Figure 13–8 shows you how to connect a secondary scan path to an SPL.

Figure 13–8. Connecting a Secondary JTAG Scan Path to an SPL



The $\overline{\text{TRST}}$ signal from the main scan path drives all devices, even those on the secondary scan paths of the SPL. The TCK signal on each target device on the secondary scan path of an SPL is driven by the SPL's DTCK signal. The TMS signal on each device on the secondary scan path is driven by the respective DTMS signals on the SPL.

DTDO on the SPL is connected to the TDI signal of the first device on the secondary scan path. DTDI on the SPL is connected to the TDO signal of the last device in the secondary scan path. Within each secondary scan path, the TDI signal of a device is connected to the TDO signal of the device before it. If the SPL is on a backplane, its secondary JTAG scan paths are on add-on boards; if signal degradation is a problem, you may need to buffer both the $\overline{\text{TRST}}$ and DTCK signals. Although less likely, you may also need to buffer the DTMS*n* signals for the same reasons.

13.9.2 Emulation Timing Calculations for SPL

The following examples help you to calculate the emulation timings in the SPL secondary scan path of your system. For actual target timing parameters, see the appropriate device data sheets.

Assumptions:

$t_{su(TTMS)}$	Target TMS/TDI setup to TCK high	10 ns
$t_{d(TTDO)}$	Target TDO delay from TCK low	15 ns
$t_{d(bufmax)}$	Target buffer delay, maximum	10 ns
$t_{d(bufmin)}$	Target buffer delay, minimum	1 ns
$t_{(bufskew)}$	Target buffer skew between two devices in the same package: $[t_{d(bufmax)} - t_{d(bufmin)}] \times 0.15$	1.35 ns
$t_{(TCKfactor)}$	Assume a 40/60 duty cycle clock	0.4 (40%)

Given in the SPL data sheet:

$t_{d(DTMSmax)}$	SPL DTMS/DTDO delay from TCK low, maximum	31 ns
$t_{su(DTDLmin)}$	DTDI setup time to SPL TCK high, minimum	7 ns
$t_{d(DTCKHmin)}$	SPL DTCK delay from TCK high, minimum	2 ns
$t_{d(DTCKLmax)}$	SPL DTCK delay from TCK low, maximum	16 ns

There are two key timing paths to consider in the emulation design:

- The TCK-to-DTMS/DTDO path, called $t_{pd(TCK–DTMS)}$, and
- The TCK-to-DTDI path, called $t_{pd(TCK–DTDI)}$.

Of the following two cases, the worst-case path delay is calculated to determine the maximum system test clock frequency.

Case 1: Single processor, direct connection, DTMS/DTDO timed from TCK low.

$$t_{pd\,(TCK–DTMS)} = \frac{\left[t_{d\,(DTMSmax)} + t_{d\,(DTCKHmin)} + t_{su\,(TTMS)}\right]}{t_{(TCKfactor)}}$$

$$= \frac{[31ns + 2ns + 10ns]}{0.4}$$

$$= 107.5ns\ (9.3\ MHz)$$

$$t_{pd\,(TCK–DTDI)} = \frac{\left[t_{d\,(TTDO)} + t_{d\,(DTCKLmax)} + t_{su\,(DTDLmin)}\right]}{t_{(TCKfactor)}}$$

$$= \frac{[15ns + 16ns + 7ns]}{0.4}$$

$$= 9.5ns\ (10.5\ MHz)$$

In this case, the TCK-to-DTMS/DTDL path is the limiting factor.

Case 2: Single/multiprocessor, DTMS/DTDO/TCK buffered input, DTDI buffered output, DTMS/DTDO timed from TCK low.

$$t_{pd\,(TCK–TDMS)} = \frac{\left[t_{d\,(DTMSmax)} + t_{(DTCKHmin)} + t_{su\,(TTMS)} + t_{(bufskew)}\right]}{t_{(TCKfactor)}}$$

$$= \frac{[31ns + 2ns + 10ns + 1.35ns]}{0.4}$$

$$= 110.9ns\ (9.0\ MHz)$$

$$t_{pd\,(TCK–DTDI)} = \frac{\left[t_{d\,(TTDO)} + t_{d\,(DTCKLmax)} + t_{su\,(DTDLmin)} + t_{d\,(bufskew)}\right]}{t_{(TCKfactor)}}$$

$$= \frac{[15ns + 15ns + 7ns + 10ns]}{0.4}$$

$$= 120ns\ (8.3\ MHz)$$

In this case, the TCK-to-DTDI path is the limiting factor.

13.9.3 Using Emulation Pins

The EMU0/1 pins of TI devices are bidirectional, three-state output pins. When in an inactive state, these pins are at high impedance. When the pins are active, they function in one of the two following output modes:

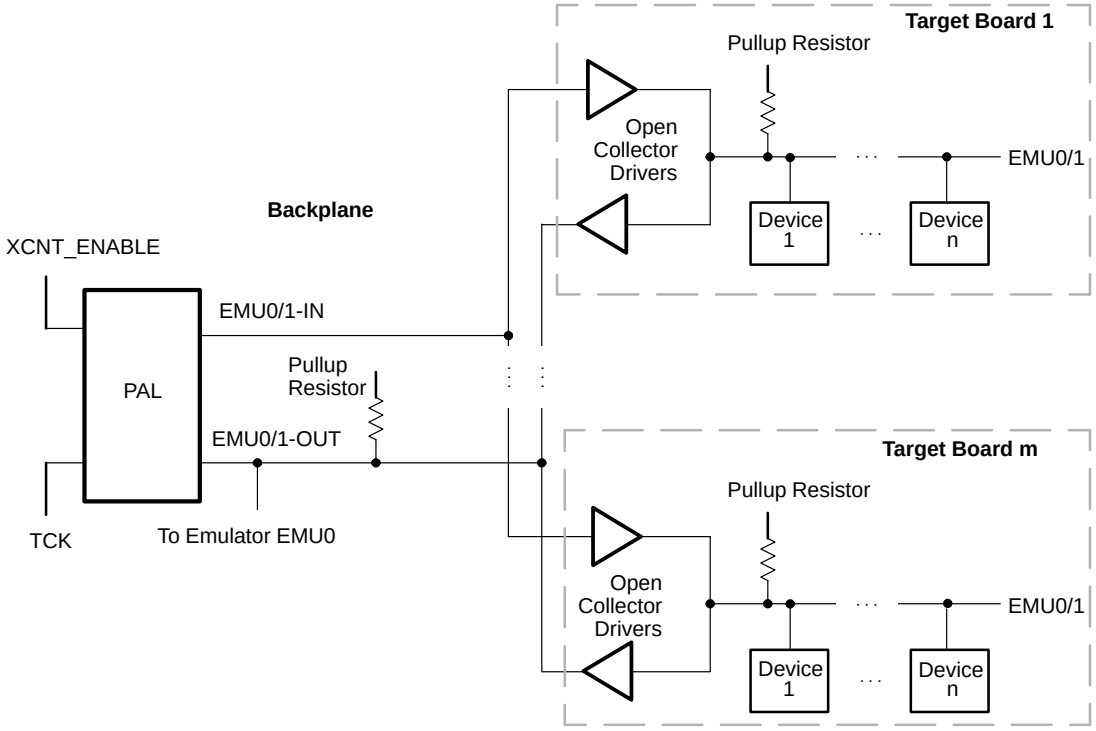
- **Signal Event**
  The EMU0/1 pins can be configured via software to signal internal events. In this mode, driving one of these pins low can cause devices to signal such events. To enable this operation, the EMU0/1 pins function as open-collector sources. External devices such as logic analyzers can also be connected to the EMU0/1 signals in this manner. If such an external source is used, it must also be connected via an open-collector source.

- **External Count**
  The EMU0/1 pins can be configured via software as totem-pole outputs for driving an external counter. If the output of more than one device is configured for totem-pole operation, then these devices can be damaged. The emulation software detects and prevents this condition. However, the emulation software has no control over external sources on the EMU0/1 signal. Therefore, all external sources must be inactive when any device is in the external count mode.

TI devices can be configured by software to halt processing if their EMU0/1 pins are driven low. This feature, in combination with the use of the signal event output mode, allows one TI device to halt all other TI devices on a given event for system-level debugging.

If you route the EMU0/1 signals between boards, they require special handling because these signals are more complex than normal emulation signals. Figure 13–9 shows an example configuration that allows any processor in the system to stop any other processor in the system. Do not tie the EMU0/1 pins of more than 16 processors together in a single group without using buffers. Buffers provide the crisp signals that are required during a RUNB (run benchmark) debugger command or when the external analysis counter feature is used.

Figure 13–9. EMU0/1 Configuration



Notes: 1) The low time on EMUx-IN should be at least one TCK cycle and less than 10 μs. Software will set the EMUx-OUT pin to a high state.

2) To enable the open-collector driver and pullup resistor on EMU1 to provide rising/falling edges of less than 25 ns, the modification shown in this figure is suggested. Rising edges slower than 25 ns can cause the emulator to detect false edges during the RUNB command or when the external counter selected from the debugger analysis menu is used.

These seven important points apply to the circuitry shown in Figure 13–9 and the timing shown in Figure 13–10:

- ❑ Open-collector drivers isolate each board. The EMU0/1 pins are tied together on each board.
- ❑ At the board edge, the EMU0/1 signals are split to provide IN/OUT. This is required to prevent the open-collector drivers from acting as a latch that can be set only once.
- ❑ The EMU0/1 signals are bused down the backplane. Pullup resistors are installed as required.
- ❑ The bused EMU0/1 signals go into a PAL® device (see Appendix A), whose function is to generate a low pulse on the EMU0/1-IN signal when

a low level is detected on the EMU0/1-OUT signal. This pulse must be longer than one TCK period to affect the devices, but less than 10 μs to avoid possible conflicts or retriggering, once the emulation software clears the device's pins.

- During a RUNB debugger command or other external analysis count, the EMU0/1 pins on the target device become totem-pole outputs. The EMU1 pin is a ripple carry-out of the internal counter. EMU0 becomes a *processor-halted* signal. During a RUNB or other external analysis count, the EMU0/1-IN signal to all boards must remain in the high (disabled) state. You must provide some type of external input (XCNT_ENABLE) to the PAL to disable the PAL from driving EMU0/1-IN to a low state.
- If sources other than TI processors (such as logic analyzers) are used to drive EMU0/1, their signal lines must be isolated by open-collector drivers and be inactive during RUNB and other external analysis counts.
- You must connect the EMU0/1-OUT signals to the emulation header or directly to a test bus controller.

Figure 13–10. Suggested Timings for the EMU0 and EMU1 Signals

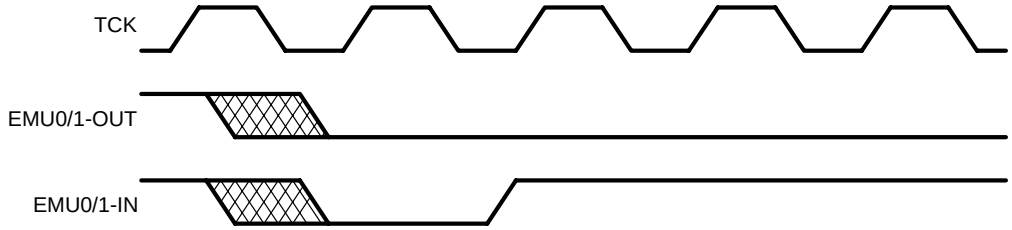
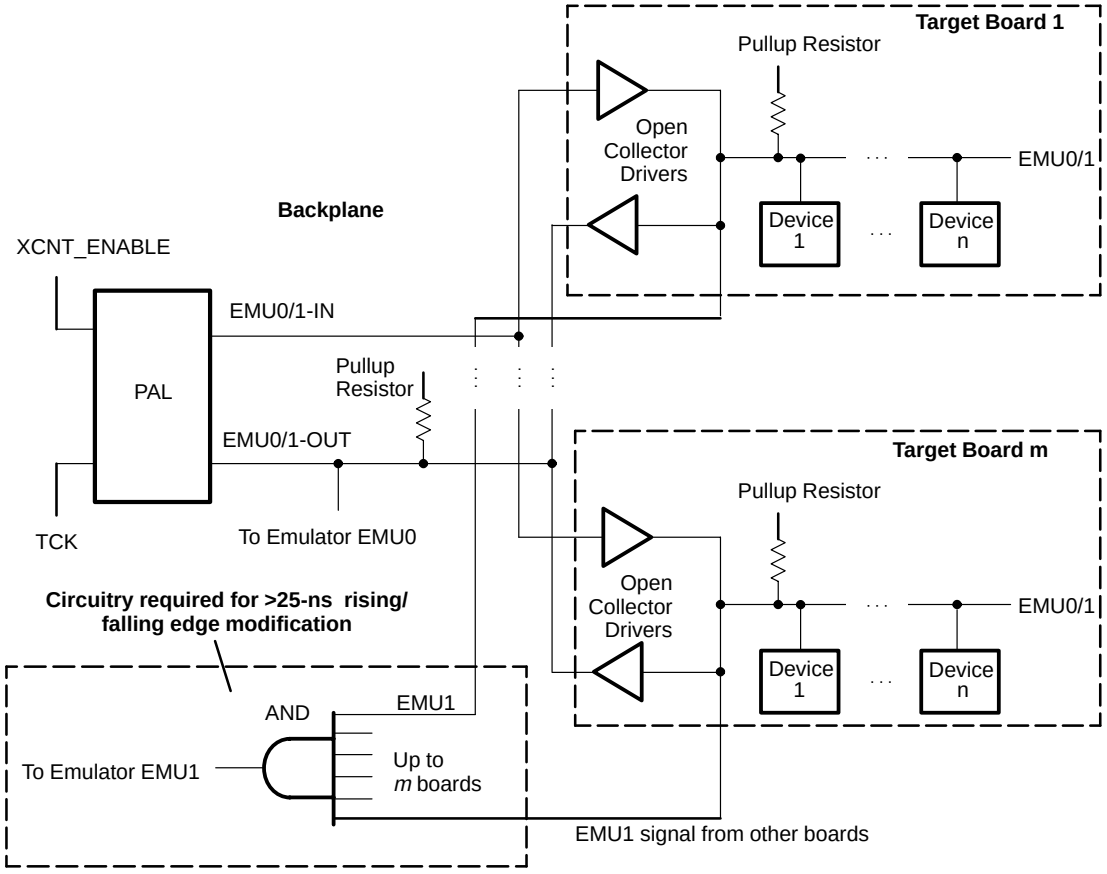


Figure 13–11. EMU0/1 Configuration With Additional AND Gate to Meet Timing Requirements

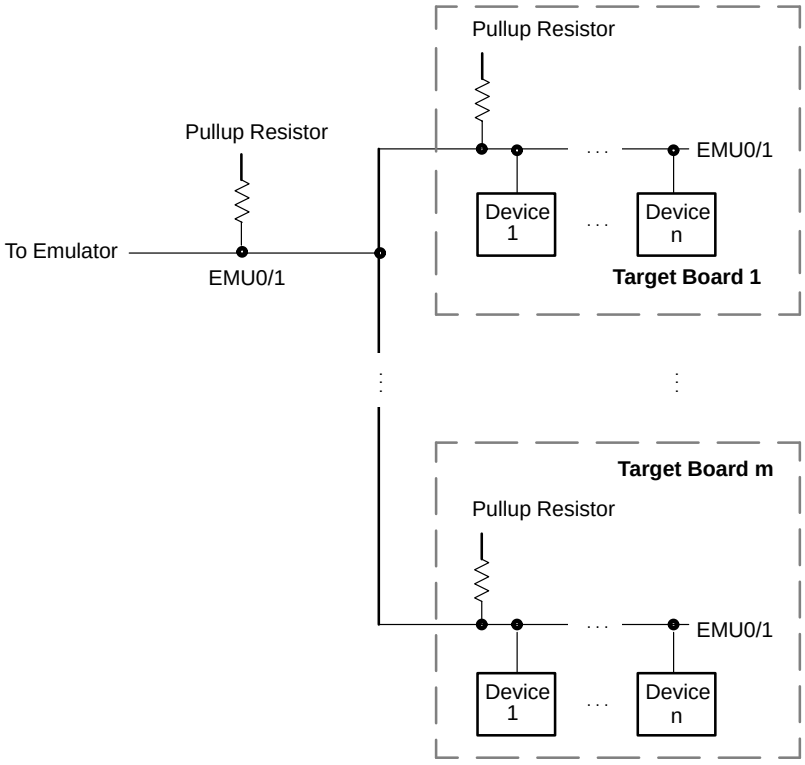


Notes: 1) The low time on EMUx–IN should be at least one TCK cycle and less than 10 μs. Software will set the EMUx–OUT pin to a high state.

2) To enable the open-collector driver and pullup resistor on EMU1 to provide rising/falling edges of less than 25 ns, the modification shown in this figure is suggested. Rising edges slower than 25 ns can cause the emulator to detect false edges during the RUNB command or when the external counter selected from the debugger analysis menu is used.

If having devices on one target board stopped by devices on another target board via the EMU0/1 signals is not important, then the circuit in Figure 13–12 can be used. In this configuration, the global-stop capability is lost. It is important not to overload EMU0/1 with more than 16 devices.

Figure 13–12. EMU0/1 Configuration Without Global Stop

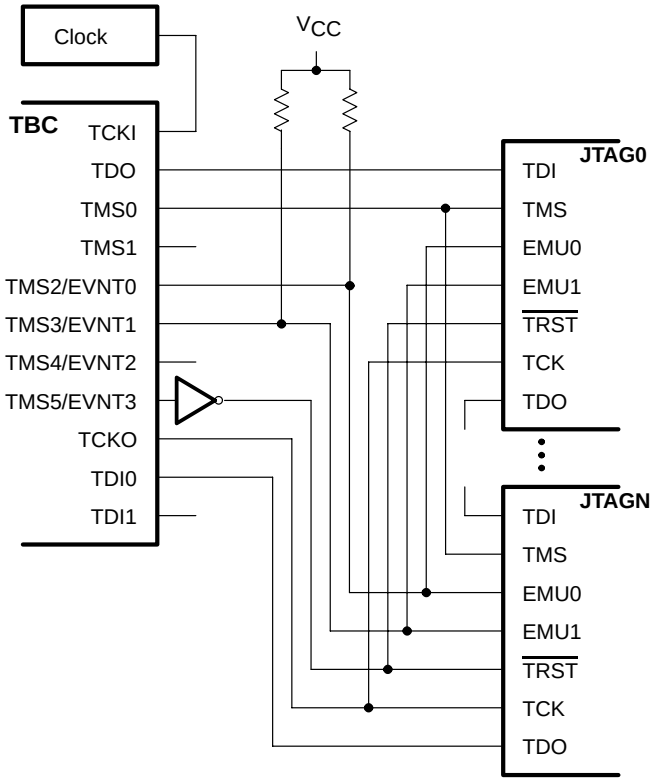


Note: The open-collector driver and pullup resistor on EMU1 must be able to provide rising/falling edges of less than 25 ns. Rising edges slower than 25 ns can cause the emulator to detect false edges during the RUNB command or when the external counter selected from the debugger analysis menu is used. If this condition cannot be met, then the EMU0/1 signals from the individual boards should be ANDed together (as shown in Figure 1-11) to produce an EMU0/1 signal for the emulator.

13.9.4 Performing Diagnostic Applications

For systems that require built-in diagnostics, it is possible to connect the emulation scan path directly to a TI ACT8990 test bus controller (TBC) instead of the emulation header. The TBC is described in the Texas Instruments *Advanced Logic and Bus Interface Logic Data Book* (literature number SCYD001). Figure 13–13 shows the scan path connections of *n* devices to the TBC.

Figure 13–13. TBC Emulation Connections for n JTAG Scan Paths



In the system design shown in Figure 1–13, the TBC emulation signals TCKI, TDO, TMS0, TMS2/EVNT0, TMS3/EVNT1, TMS5/EVNT3, TCKO, and TDI0 are used, and TMS1, TMS4/EVNT2, and TDI1 are not connected. The target devices' EMU0 and EMU1 signals are connected to V_{CC} through pullup resistors and tied to the TBC's TMS2/EVNT0 and TMS3/EVNT1 pins, respectively. The TBC's TCKI pin is connected to a clock generator. The TCK signal for the main JTAG scan path is driven by the TBC's TCKO pin.

On the TBC, the TMS0 pin drives the TMS pins on each device on the main JTAG scan path. TDO on the TBC connects to TDI on the first device on the main JTAG scan path. TDI0 on the TBC is connected to the TDO signal of the last device on the main JTAG scan path. Within the main JTAG scan path, the TDI signal of a device is connected to the TDO signal of the device before it. $\overline{\text{TRST}}$ for the devices can be generated either by inverting the TBC's TMS5/EVNT3 signal for software control or by logic on the board itself.

Chapter 14

External Signal Description

This chapter describes external signals used by the CPU to communicate with memory, peripherals, and external devices.

Table 14–1. External Signal Description

(a) Clock / PLL

Name	Type	Description
CLKIN	I	Clock Input
CLKOUT1	O	Clock output at full device speed
CLKOUT2	O	Clock output at half of device speed
CLKMODE(1:0)	I	Clock mode select Selects whether the output clock frequency = input clock frequency x4 or x1
PLLFREQ(3:1)	I	PLL Frequency Range Selects one of 5 frequency ranges bounding the CLKOUT1 signal CLKOUT1 frequency determines the 3-bit value for the PLLFREQ pins.
PLLV	A†	PLL analog VCC connection for the low-pass filter
PLLG	A†	PLL analog GND connection for the low-pass filter
PLLF	A†	PLL low-pass filter connection to external components and a bypass capacitor

† A = Analog Signal (PLL Filter)

(b) **J***TAG Emulation*

Name	Type	Description
TMS	I	JTAG test port mode select (features an internal pull-up)
TDO	O/Z	JTAG test port data out
TDI	I	JTAG test port data in (features an internal pull-up)
TCK	I	JTAG test port clock
$\overline{\text{TRST}}$	I	JTAG test port reset (features an internal pull-down)
EMU1	I/O/Z	Emulation pin 1, pull-up with a dedicated 20KΩ resistor
EMU0	I/O/Z	Emulation pin 0, pull-up with a dedicated 20KΩ resistor

Table 14–1. External Signal Description (Continued)

(c) Control

Name	Type	Description
$\overline{\text{RESET}}$	I	Device Reset
NMI	I	Non-maskable interrupt
EXT_INT(7:4)	I	External interrupts
IACK	O	Interrupt acknowledge for all active interrupts serviced by the CPU
INUM(3:0)	O	Active interrupt identification number Valid during IACK for all active interrupts (not just external) Encoding order follows the Interrupt Service Fetch Packet ordering
LENDIAN	I	If high, selects Little Endian byte/half-word addressing order within a word If low, selects Big Endian addressing
$\overline{\text{PD}}$	O	Power-down mode2 or 3 active if low
BOOTMODE(4:0)	I	Boot Mode

(d) Host Port Interface (HPI)

Name	Type	Description
$\overline{\text{HINT}}$	O/Z	Host Interrupt (from DSP to Host)
HCNTRL(1:0)	I	Host control – selects between Control, Address or Data registers
HHWIL	I	Host halfword select – first or second halfword (not necessarily high or low order)
$\overline{\text{HBE1}}$	I	Host byte select within word or half-word
$\overline{\text{HBE0}}$	I	Host byte select within word or half-word
$\overline{\text{HR/W}}$	I	Host read or write select
HD(15:0)	I/O/Z	Host Port Data (used for transfer of data, address and control)
$\overline{\text{HAS}}$	I	Host address strobe
$\overline{\text{HCS}}$	I	Host chip select
$\overline{\text{HDS1}}$	I	Host data strobe 1
$\overline{\text{HDS2}}$	I	Host data strobe 2
$\overline{\text{HRDY}}$	O	Host ready (from DSP to Host)

Table 14–1. External Signal Description (Continued)

(e) External Memory Interface (EMIF)
EMIF – Control Signals Common to All Types of Memory

Name	Type	Description
$\overline{\text{CE}}$(3:0)	O/Z	Memory space enables Enabled by bits 24 and 25 of the word address Only one asserted during any external data access
$\overline{\text{BE}}$(3:0)	O/Z	Byte enable control (decoded from the 2 lowest bits of the internal address). Byte write enables for most types of memory. Can be directly connected to SDRAM read and write mask signal (SDQM)
EA(21:2)	O/Z	External Address (word address)
ED(31:0)	I/O/Z	External Data

(f) EMIF – Asynchronous Memory Control

Name	Type	Description
$\overline{\text{ARE}}$	O/Z	Asynchronous Memory read strobe
$\overline{\text{AOE}}$	O/Z	Asynchronous Memory output enable
$\overline{\text{AWE}}$	O/Z	Asynchronous Memory write enable
ARDY	I	Asynchronous Memory ready signal

(g) EMIF – Synchronous Burst SRAM Control

Name	Type	Description
$\overline{\text{SSADS}}$	O/Z	SBSRAM address strobe
$\overline{\text{SSOE}}$	O/Z	SBSRAM output enable
$\overline{\text{SSWE}}$	O/Z	SBSRAM write enable
SSCLK	O/Z	SBSRAM clock

Table 14–1. External Signal Description (Continued)

(h) EMIF – Synchronous DRAM Control

Name	Type	Description
SDA10	O/Z	SDRAM address 10 (separate for refresh)
$\overline{\text{SDRAS}}$	O/Z	SDRAM row address strobe
$\overline{\text{SDCAS}}$	O/Z	SDRAM column address strobe
$\overline{\text{SDWE}}$	O/Z	SDRAM write enable
SDCLK	O/Z	SDRAM clock

(i) EMIF – Bus Arbitration

Name	Type	Description
$\overline{\text{HOLD}}$	I	Hold request from the host
$\overline{\text{HOLDA}}$	O	Hold request acknowledge to the host

(j) Timers

Name	Type	Description
TOUT1	O/Z	Timer 1 or general purpose output
TOUT0	O/Z	Timer 0 or general purpose output
TINP1	I	Timer 1 or general purpose input
TINP0	I	Timer 0 or general purpose input

(k) DMA

Name	Type	Description
DMAC(3:0)	O	DMA Action Complete

Table 14–1. External Signal Description (Continued)

(l) Multi Channel Serial Port 1 (MCSP1)

Name	Type	Description
CLKS1	I	External clock source
CLKR1	I/O/Z	Receive clock
CLKX1	I/O/Z	Transmit clock
DR1	I	Receive data
DX1	O/Z	Transmit data
FSR1	I/O/Z	Receive frame sync
FSX1	I/O/Z	Transmit frame sync

(m) Multi Channel Serial Port 0 (MCSP0)

Name	Type	Description
CLKS0	I	External clock source
CLKR0	I/O/Z	Receive clock
CLKX0	I/O/Z	Transmit clock
DR0	I	Receive data
DX0	O/Z	Transmit data
FSR0	I/O/Z	Receive frame sync
FSX0	I/O/Z	Transmit frame sync

Appendix A

Glossary

A

access: Generic term which includes memory load, and/or memory store.

address: The location of program code or data stored; an individually accessible memory location.

ALU: See *arithmetic logic unit*.

application-specific integrated circuit: A custom chip designed for a specific application. It is designed by integrating standard cells from a library.

arithmetic logic unit (ALU): The hardware of the CPU that performs arithmetic and logic function.

B

block: The three least significant bits of the program address. These correspond to the address within a fetch packet of the first instruction being addressed.

boot: The process of loading a program into memory.

boot configuration: A set of parameters defining how a device is booted.

boot loader: A built-in segment of code that transfers code from an external source to program memory at power up.

C

cache: A fast storage buffer in the central processing unit of a computer.

cache controller: Coordinates program accesses between CPU program fetch mechanism and the cache as well as external memory.

central processing unit (CPU): The unit that coordinates the functions of a processor.

circular addressing: An address mode in which a finite set of addresses is reused by linking the largest address back to the smallest address.

clock cycles: A periodic or sequence of events based on the input from the external clock.

code: A set of instructions written to perform a task; a computer program or part of a program.

compiler: A computer program that translates programs in a high-level language into their assembly-language equivalents.

control register: A register that contains bit fields which define the way a device operates.

control register file: A set of control registers.

CPU: See *central processing unit.*

crosspath: A link between register files to provide communication between the CPU units.

D

data memory: Memory accessed through the ‘C62xx’s RAM interface.

data memory controller: Coordinates RDY-stalls and memory requests between the ’C6xx load/store units, and internal/external memory.

DMA: Direct Memory Access.

DMA destination: The module where the DMA data ends up. DMA data is written to the DMA destination.

DMS I/F: One of 5 possible sources/destinations (H/W which interfaces the DMA controller to the proper memory, based on memory address).

DMA operation: Generic term referring to a series of DMA data transfers.

DMA source: The module where the DMA data originates. DMA data is read from the DMA source.

DMA transfer: Generic term referring to data transferred from one part of memory to another. Each DMA transfer consists of a read bus cycle (source –> DMA holding register), and a write bus cycle (DMA holding register –> destination).

DMS: Data Memory System

dynamic random access memory (DRAM): Memory that can be read and written by the microprocessor and whose storage locations can be accessed in any order but must be refreshed (recharged) periodically to retain data or program code.

E

E1: A European high-speed network communication service that operates at 2.048M bits per second and uses A-law companding.

erasable programmable read-only memory (EPROM): A memory device whose contents are erasable (usually via UV light), and programmable.

external memory interface (EMIF): Microprocessor hardware which is used to read to and write from off-chip memory.

F

fetch packet: A contiguous eight word series of instructions fetched by the CPU. Aligned on an eight word boundary.

first-in, first-out: A method for managing a set of items to which additions and deletions are made; items are added to one end of the list and removed from the other.

fixed-point processor: A processor which does arithmetic operations using integer arithmetic with no exponents.

Flash memory: Electronically erasable, programmable nonvolatile (read-only) memory.

floating-point processor: A processor capable of handling floating-point arithmetic where real operands are represented using exponents.

frame: An eight word space in the cache RAMs. Each fetch packet in the cache resides in only one frame. A cache update will load a frame with the requested fetch packet. The cache contains 512 frames.

G

global interrupt enable bit (GIE): A bit in the control status register (CSR) that is used to enable or disable maskable interrupts.

H

half-word: For this device, a half-word is taken as a 16-bit data item taken as a unit.

hardware interrupt: An interrupt triggered through physical connections with on-chip peripherals or external devices.

host port interface (HPI): A parallel interface that the CPU uses to communicate with a host processor.

I

index: A nine-bit field in the program address which specifies which of the 512 frames in the cache the current access is mapped into.

indirect addressing: Indirect addressing: An addressing mode in which an address points to another pointer rather than to the actual data; this mode is prohibited in RISC architecture.

instruction fetch packet: A group of up to eight instructions held in memory for execution by the CPU.

interrupt: A signal sent by hardware or software to a processor requesting attention. An interrupt tells the processor to suspend its current operation, save the current task status, and perform a particular set of instructions. Interrupts communicate with the operating system and prioritize tasks to be performed.

interrupt service fetch packet (ISFP): A fetch packet used to service interrupts. If eight instructions are insufficient, the user must branch out of this block for additional interrupt service. If the delay slots of the branch do not reside within the ISFP, execution continues from execute packets in the next fetch packet (the next ISFP).

L

latency: The delay between the occurrence of a condition and the reaction of the device. Also, in a pipeline, the necessary delay between the execution of two instructions to ensure that the values used by the second instruction are correct.

least significant bit (LSB): The lowest order bit in a word.

M

maskable interrupt: A hardware interrupt that can be enabled or disabled through software.

million instructions per second (MIPS): A measure of the execution speed of a computer.

most significant bit (MSB): The highest order bit in a word.

multichannel buffered serial port (McBSP): An on-chip full-duplex circuit that provides direct serial communication through several channels to external serial devices.

multiplexer: A device for selecting one of several available signals.

multiplier: A CPU component that multiplies the contents of two registers.

multivendor internet protocol: A standard network protocol supported by several major network communication vendors.

N

nonmaskable interrupt (NMI): An interrupt that can be neither masked nor disabled.

normalization: The reduction of a complex data structure to its simplest form or of a circuit to its lowest number of gates.

O

overflow: A condition in which the result of an arithmetic operation exceeds the capacity of the register used to hold that result.

P

packing: Minimizing the space occupied by data or memory through the elimination of discontinuous spaces between segments.

parallelism: Sequencing events to occur simultaneously. Parallelism is achieved in a CPU by using instruction pipelining.

peripheral: A device connected to and usually controlled by a host device.

pipeline: A method of executing instructions in which the output of one process serves as the input to another, much like an assembly line. These processes become the stages or phases of the pipeline.

pipeline processing: A technique that provides simultaneous, or parallel, processing within the computer. It refers to overlapping operations by moving data or instructions into a conceptual pipe with all stages of the pipe processing simultaneously.

phase-locked loop: A circuit for synchronizing a variable oscillator with the phase of the transmitted signal.

program cache: A fast memory cache for storing program instructions allowing for quick execution.

program address: A 30-bit value placed on the PADDR lines of the CPU boundary which specifies the address of the current requested program data packet.

program memory: Memory accessed through the ‘C62xx’s program fetch interface.

R

random-access memory (RAM): A type of memory device in which the individual locations can be accessed in any order.

register: A small area of high speed memory, located within a processor or electronic device, that is used for temporarily storing data or instructions. Each register is given a name, contains a few bytes of information, and is referenced by programs.

reduced-instruction set computer (RISC): A computer whose instruction set and related decode mechanism are much simpler than those of micro-programmed complex instruction set computers. The result is a higher instruction throughput and a faster real-time interrupt service response from a smaller, cost-effective chip.

reset: A means of bringing the CPU to a known state by setting the registers and control bits to predetermined values and signaling execution to start at a specified address.

S

saturation: A state where any further input no longer results in the expected output.

synchronous burst static random-access memory (SBSRAM): RAM whose contents does not have to be refreshed periodically. Transfer of data is at a fixed rate relative to the clock speed of the device, but the speed is increased.

synchronous dynamic random-access memory (SDRAM): RAM whose contents is refreshed periodically, or the data is lost. Transfer of data is at a fixed rate relative to the clock speed of the device.

shifter: A hardware unit that shifts bits in a word to the left or to the right.

SBSRAM: synchronous burst SRAM

T

tag: The eighteen most significant bits of the program address. This value corresponds to the physical address of the fetch packet that is in that frame.

W

word: A 32-bit value.

Index

H

I

J

M

O

P

R

S

T

X

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.